



**Universidad
Zaragoza**

Trabajo Fin de Grado

RECONOCIMIENTO AUTOMÁTICO DE TIPOS DE DEPENDENCIAS BASADO EN TÉCNICAS DE DEEP LEARNING

Autor/es

Kevin Cedeño Gonzáles

Director/es

Emiliano Bernués Del Río

Escuela de Ingeniería y Arquitectura
2015

Agradecimientos

Este trabajo de fin de grado es un punto y seguido a una importante etapa de mi vida, esperando hacer justicia a las aportaciones de todas las personas que han influido tanto directa como indirectamente en la realización el mismo.

Primero que todo quiero dar las gracias a Emiliano, mi director del trabajo, que ha sido paciente y comprensivo a la hora de orientarme en el desarrollo del trabajo y la redacción de esta memoria, sin él no hubiera sido posible, muchas gracias Emiliano.

Igualmente a Miguel y Mario que durante el verano me han ayudado y han compartido el tiempo conmigo.

Dar las gracias también a los profesores que me han instruido durante el grado, espero que en mí se reflejen las mejores cualidades de cada uno de ellos.

No quisiera olvidar a mi madre, mi padre y mi hermana quienes siempre se han sacrificado y me han apoyado durante toda mi vida, os quiero.

Por último y casi lo más importante a mis amigos, a todos y cada uno de mis compañeros que me han acompañado en este camino, que han apartado las piedras que han aparecido y tendiéndome la mano en los momentos que he tropezado, en especial a Raquel, David, Nacho y Fabio que han sido los pilares fundamentales de mis días durante estos años, que son la familia que se puede elegir y que estoy muy orgulloso de haber elegido, espero que no dejemos de vernos aunque los kilómetros nos separen.

Reconocimiento automático de tipos de dependencias basado en técnicas de deep learning

Resumen

Ayudar a personas que sufren enfermedades degenerativas sin solución a día de hoy es un problema abarcado en todos los ámbitos de la ciencia; en el campo de la medicina es una labor que conlleva demasiado tiempo y esfuerzo obtener una cura, sin embargo la ingeniería es un campo que puede proporcionar una mayor calidad de vida al paciente.

En este trabajo de fin de grado se ha realizado un sistema automático capaz de asistir a personas con enfermedades de este tipo, más concretamente de Alzheimer, reconociendo la habitación en la que se encuentra la persona asistida ante la posibilidad de la pérdida de memoria o desorientación que puede producir dicha afección.

El sistema se entrenará siguiendo una de las técnicas con mayor potencial en la actualidad, denominada deep learning. El sistema, por tanto, tomará una estructura concreta llamada red neuronal convolucional.

Para ello se ha realizado un estado del arte donde se ha revisado las características de este proceso y la actual práctica que se tiene con este tipo de técnicas dando importancia a la comprensión de cada paso que se realiza.

Además se ha llevado a cabo la implementación de la red neuronal así como todo lo necesario para poder realizar el entrenamiento de la forma más completa posible.

Se ha estudiado posibles errores que se puedan producir durante el entrenamiento así como un comportamiento normal en él de forma que se pueda ahorrar tiempo y esfuerzo ante la presencia de alguna acción poco convencional.

Por último, se ha realizado un estudio de los resultados obtenidos así como una línea de futuro proponiendo mejoras que podrían ofrecer una mayor contribución al paciente.



DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./D^a. _____,

con nº de DNI _____ en aplicación de lo dispuesto en el art.

14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster)
_____, (Título del Trabajo)

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, _____

Kevin Cortés

Fdo: _____

Índice general

Introducción	11
Organización de la memoria	13
Estudio del arte	16
Conceptos básicos	16
Algoritmo back-propagation.....	19
Arquitectura utilizada	23
Tipos de subcapas.....	24
Cálculos en GPU.....	28
Implementación y herramientas	29
Desarrollo del trabajo.....	31
Base de datos	31
Data augmentation	32
Programación de la red neuronal	34
Pruebas de entrenamiento y posibles problemas.....	39
Sustitución de imágenes.....	39
Sustracción de la media de la base de datos	41
Número de imágenes	43
Entrenamiento de la red neuronal convolucional y resultados	44
Estructura del programa.....	54
Programa de descarga de imágenes en Java.....	54
Programa de etiquetado y construcción de la base de datos en C++.....	55
Programas de acondicionamiento de la base de datos y entrenamiento de la red neuronal convolucional en MATLAB	57
Conclusiones y líneas futuras.....	60
Conclusiones.....	60
Líneas futuras	61
Referencias.....	62
Anexo 1. Programación de descarga de imágenes en Bing utilizando Java	64

Anexo 2. Programación de creación y etiquetado de una base de datos.....	67
Anexo 3. Creación de código para aprovechamiento de un código ejemplo.....	69
Anexo 4. Formato de una red neuronal en MATLAB	71
Anexo 5. Creación de código para realizar data augmentation	73

Índice de figuras

<i>Figura 1. Partes de una neurona</i>	<i>16</i>
<i>Figura 2. Estructura de una perceptrón.....</i>	<i>17</i>
<i>Figura 3. Entrenamiento de red de cuatro capas, con ReLU en la línea continua y con la función tanh en la línea discontinua [2]</i>	<i>19</i>
<i>Figura 4. Diagrama básico de red neuronal</i>	<i>20</i>
<i>Figura 5. Capa de entrada y primera capa oculta de la red neuronal, los cuadrados negros representan los filtros y los azules las imágenes.....</i>	<i>26</i>
<i>Figura 6. Segunda, tercera y cuarta capa oculta.....</i>	<i>26</i>
<i>Figura 7. Fase de clasificación de la red neuronal.....</i>	<i>27</i>
<i>Figura 8. Data augmentation aplicado a una de las imágenes.....</i>	<i>33</i>
<i>Figura 9. Representación imágenes escogidas aleatoriamente de la base de datos.....</i>	<i>35</i>
<i>Figura 10. Representación de los filtros de la capa de entrada de la red.</i>	<i>35</i>
<i>Figura 11. Evolución del porcentaje de error en entrenamiento y en validación.....</i>	<i>36</i>
<i>Figura 12. Evolución del error cuadrático medio.....</i>	<i>36</i>
<i>Figura 13. Salida de los filtros en la capa de entrada para la letra v.</i>	<i>37</i>
<i>Figura 14. Salida de la subcapa ReLU, activación de las unidades.</i>	<i>38</i>
<i>Figura 15. Predicción del carácter.</i>	<i>38</i>
<i>Figura 16. Evolución de porcentaje de error para un conjunto de muchas imágenes artificiales aleatorias</i>	<i>40</i>
<i>Figura 17. Convergencia de los filtros de la capa de entrada para varias entradas de imágenes aleatorias</i>	<i>40</i>
<i>Figura 18. Imágenes medias de la base de datos para cada canal.....</i>	<i>41</i>
<i>Figura 19. Representación de la iniciación del filtro y de las 3 primeras iteraciones.....</i>	<i>42</i>
<i>Figura 20. Evolución de la probabilidad de error de una red no funcional</i>	<i>43</i>
<i>Figura 21. Evolución de la probabilidad de error con un conjunto muy pequeño de 1000 imágenes.....</i>	<i>44</i>
<i>Figura 22. Distribución por clases de las imágenes de la base de datos.....</i>	<i>45</i>
<i>Figura 23. Asignación de las imágenes a entrenamiento o validación.</i>	<i>45</i>
<i>Figura 24. Fase de extracción de características.....</i>	<i>46</i>

<i>Figura 25. Fase de clasificación.</i>	<i>46</i>
<i>Figura 26. Representación de 20 imágenes aleatorias pertenecientes a la base de datos.</i>	<i>47</i>
<i>Figura 27. Evolución de la probabilidad de error y del error cuadrático medio en 32 iteraciones</i>	<i>48</i>
<i>Figura 28. Representación de los filtros de la capa de entrada tras 32 iteraciones de entrenamiento</i>	<i>49</i>
<i>Figura 29. Representación de la salida de 60 filtros de los 96 pertenecientes a la capa de entrada de la imagen de un baño.....</i>	<i>49</i>
<i>Figura 30. Predicción del sistema para una imagen de un baño una vez entrenada la red neuronal</i>	<i>50</i>
<i>Figura 31. Comparación de distribuciones de las muestras en cada forma de realizar la resta</i>	<i>51</i>
<i>Figura 32. Diagrama de bloques básico del programa de descarga automática en Java.</i>	<i>54</i>
<i>Figura 33. Diagrama de bloques de la creación de la base de datos con C++.</i>	<i>55</i>
<i>Figura 34. Captura de la base de datos en gestor gráfico.....</i>	<i>56</i>
<i>Figura 35. Diagrama de bloques del acondicionamiento de la base de datos.....</i>	<i>57</i>
<i>Figura 36. Diagrama de bloques del código que entrena la red neuronal.</i>	<i>58</i>

Introducción

La búsqueda de nuevos tratamientos a enfermedades que a día de hoy no tienen solución, debido a la complejidad de la misma, es un campo que requiere gran esfuerzo y tiempo para poder alcanzar nuevos hitos en la medicina. Por ello, hoy en día se dedica empeño en otros ámbitos de la ciencia, como la ingeniería, para desarrollar nuevos métodos que puedan ayudar a la persona afectada a convivir con la enfermedad y poder darle una mejor calidad de vida.

Una de estas enfermedades es el Alzheimer cuyas consecuencias son el deterioro del intelecto y de la memoria además de la capacidad de realizar normalmente cualquier actividad diaria teniendo un alto impacto físico, psicológico y social en las personas que sufren esta enfermedad.

La tendencia a olvidar acontecimientos que han sucedido recientemente así como la desubicación en tiempo y espacio del individuo, incluso en lugares conocidos como su propio hogar, son signos claros de la degeneración que produce esta enfermedad.

Este trabajo de fin de grado se centrará en la ayuda a personas que sufran esta enfermedad buscando poder indicar la ubicación del paciente dentro de un lugar cerrado como es su domicilio, de forma que se le indique dónde está para poderle sugerir o hacerle recordar una actividad, según sus patrones de conducta previamente estudiados.

El objetivo de este trabajo será realizar un sistema capaz de captar una imagen desde el punto de vista del paciente y que se pueda reconocer la habitación en la que se encuentra procesando dicha imagen y sin que el sistema haya visto antes la imagen en cuestión.

Existen métodos de procesamiento de imagen cuya complejidad no permitía su utilización debido al elevado coste computacional que suponía, pero con la progresión del hardware de las tarjetas gráficas, impulsada por los videojuegos, hace que actualmente se lleven a cabo operaciones en paralelo de forma muy rápida y eficiente.

Este hecho, en concreto, ha llevado a que actualmente surja un impulso en el uso de sistemas basados en el procesamiento paralelo de datos, como lo son las redes neuronales convolucionales. En este trabajo se van a utilizar dichas redes las cuales, tras un exhaustivo entrenamiento, serán capaces de estudiar la composición de la fotografía

de aprendizaje de características similares, y con ellas ser capaces de reconocer tipos de imágenes.

Para poder realizar este sistema es necesario generar un conjunto de datos a partir de imágenes encontrados en Internet, mediante un buscador, en este caso Bing. Se buscarán todas las habitaciones existentes en una casa. Dichas imágenes se descargarán automáticamente a través de un programa Java, creado específicamente para esta labor.

Tras la descarga de todas las imágenes se procederá al descarte de las que no sean útiles para el entrenamiento del sistema, posteriormente se obtendrán más imágenes partiendo de variación de la imagen original como son escalado, selección aleatoria de porciones de la imagen, inversión con respecto al eje horizontal y rotación, técnica llamada data augmentation, dotando así de robustez a la red neuronal ante estas posibles variaciones en la imagen.

Tras este pre-procesado se construirá la base de datos que utilizará la red neuronal basada en una estructura dada por válida tras su estudio y resultados en la clasificación de objetos en una base de datos de imágenes llamada Imagenet [1], [2]. La base de datos se creará utilizando dos librerías OpenCV y SQLite, ambas compatibles con C++ y la última además con MATLAB pudiéndose encontrar en [3] y [4] respectivamente.

Para la creación de la red neuronal y su posterior entrenamiento, se utilizarán la librería de algoritmos de visión por computador VLFeat desarrollada en un toolbox implementado en MATLAB llamado Matconvnet [5]. El uso de esta librería aprovecha los recursos de las tarjetas gráficas para la paralelización de los cálculos, por ello se necesita una tarjeta gráfica de la marca nVidia y el toolbox CUDA [6] para poder utilizar las funciones implementadas en dicha librería.

Las clases que compondrán la base de datos serán las diferentes habitaciones que se pueden encontrar en una casa, a saber, cocinas, baños, salas de estar, dormitorios, comedores, pasillos, salas de juegos.

Después del entrenamiento, se estudiarán los resultados obtenidos siguiendo un razonamiento desde el punto de vista del error en cada iteración del entrenamiento (época), así como el resultado final de la red y su composición ya entrenada.

A continuación se presenta el contenido de los diferentes apartados que compondrán la memoria:

Organización de la memoria

- **Introducción:** Contiene una breve descripción del trabajo así como los objetivos a conseguir.
- **Estado del arte:** se realiza una aproximación a los conceptos teóricos utilizados en el trabajo así como todo lo necesario para llevarlo a cabo.
- **Desarrollo del trabajo:** se detallan las actividades que se han realizado en el trabajo para poder conseguir los objetivos propuestos estando incluidos también los problemas encontrados durante las pruebas y los resultados obtenidos.
- **Estructura del programa:** se indica como se ha procedido en la implementación de los programas necesarios para llevar a cabo la realización de este trabajo con una explicación del funcionamiento de cada parte junto a un diagrama de bloques que busca facilitar la comprensión del mismo.
- **Conclusiones y líneas futuras:** se describen las conclusiones sacadas de este trabajo así como los posibles caminos a seguir para mejorarlo y conseguir mayores propósitos.
- **Referencias**
- **Anexo 1. Programación de descarga de imágenes en Bing utilizando Java:** se explica con mayor detalle aspectos destacables en la programación de dicho programa que no se incluye en el apartado de estructura del programa ya que sería contrario a la visión superficial que se intenta dar para una comprensión intuitiva.
- **Anexo 2. Programación de creación y etiquetado de una base de datos:** siguiendo el propósito del Anexo 1, se dan detalles de relevancias de la programación de la base de datos.
- **Anexo 3. Creación de código para aprovechamiento de un código ejemplo:** se hace una diferenciación de las partes de código que han sido creadas para el trabajo y las que ya se disponían para el funcionamiento del sistema.
- **Anexo 4. Formato de una red neuronal en MATLAB:** se presenta la estructura que sigue una red neuronal en MATLAB para poder utilizarla.

- **Anexo 5. Creación de código para realizar data augmentation:** se explica las acciones más interesante del procesado de las imágenes para realizar la técnica de data augmentation explicada y utilizada en este trabajo.

Estudio del arte

Uno de los métodos de procesamiento de datos que está ganando fuerza hoy en día es el uso de redes neuronales convolucionales complejas de elevado número de neuronas y capas, denominado deep learning para una multitud de usos en varios sectores como el médico, en sistemas multimedia, en defensa o incluso en redes sociales.

Conceptos básicos

Para explicar qué es y cómo se comporta una red neuronal se introduce el concepto de perceptrón, modelo matemático creado en los años 60 por Frank Rosenblatt [7], el cual reproduce el comportamiento del procesamiento de la información de una neurona a través de expresiones matemáticas.

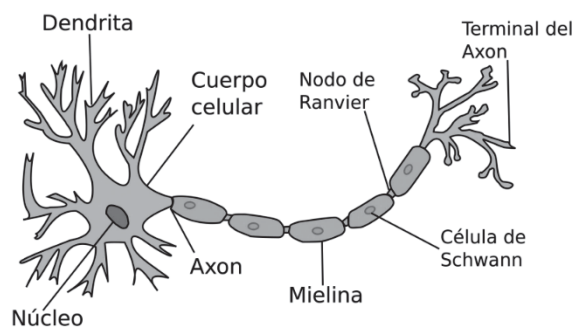


Figura 1. Partes de una neurona

Se expresa la llegada de la información por las entradas de la neurona llamadas dendritas y modulada en el núcleo de la neurona para su posterior salida hacia las otras neuronas interconectadas a través del axón; tenemos entonces un modelo (ver Figura 2) en el que se obtiene una salida a partir de varias entradas multiplicadas por unas constantes, pesos, y sumadas; la suma pasará por una función para realizar la decisión del valor de salida llamada función de activación, la cual nos indicará cuando se estimula una neurona dada.

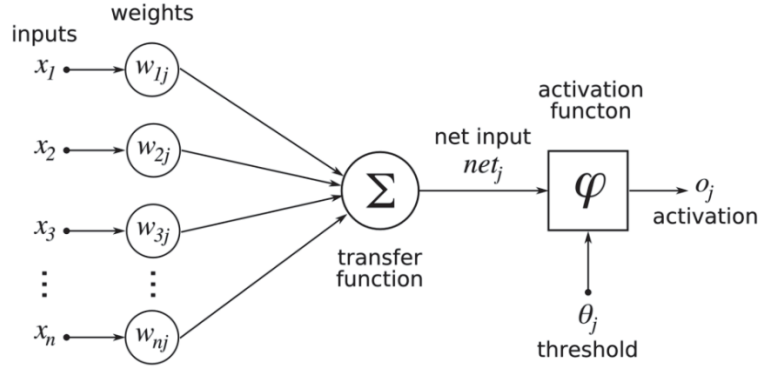


Figura 2. Estructura de una perceptrón

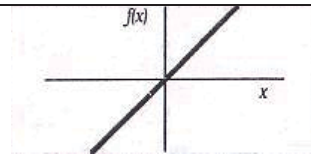
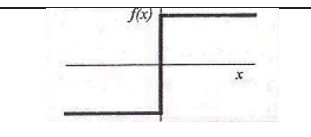
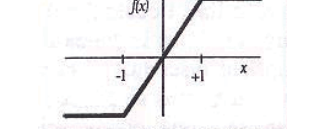
Si agrupamos varias neuronas y a cada agrupación la llamamos capas, la unión de varias capas genera lo que se conoce como un perceptrón multicapa, donde, en cada capa se obtendrán características singulares para cada imagen, esta es la base sobre la que se sustentan las redes neuronales.

La fórmula matemática, de la Figura 2, asociada al perceptrón será

$$o_j = f \left(\sum_{i=1}^N x_i \cdot w_{ij} + b_j \right) \quad (1)$$

Donde la función f será la función de activación de la neurona, x_i las muestras de entrada de la señal, w_{ij} los pesos y o_j la salida de la misma.

La función de activación recogerá la estimulación de una neurona según su argumento, toma valores dependiendo del valor de su entrada de forma que sirva como umbral; según su forma se puede clasificar en seis tipos, recogidas en la siguiente tabla:

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, \infty]$	
Escalón	$y = \text{sign}(x)$	$[-1, +1]$	
Lineal	$y = \begin{cases} -1, & \text{si } x < -l \\ x, & \text{si } -l \leq x \leq l \\ 1, & \text{si } x > l \end{cases}$	$[-1, +1]$	

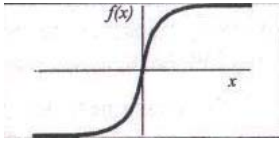
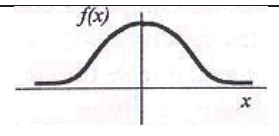
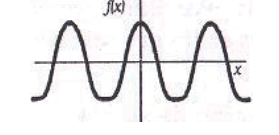
Sigmoide	$y = \frac{1}{1 + e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0, +1]$	
Sinusoidal	$y = A\sin(\omega x + \varphi)$	$[-1, +1]$	

Tabla 1. Funciones de activación típicas utilizadas en redes neuronales

Para la extracción de las características de los diferentes objetos que componen la escena, las transformaciones lineales que se suelen usar para el tratamiento de las señales, como la transformada de Fourier, no es suficiente para abordar este tipo de problemática de visión por computador donde es de gran importancia la localización espacial de la información en la señal.

Por tanto, de las funciones de activación típicas mostradas en la tabla anterior, se suele usar la función sigmoide teniendo una salida saturada y dando menor importancia a los valores más bajos siendo válidas cualquiera de las dos expresiones; para mejorar el tiempo de entrenamiento de la red neuronal, se sigue el estudio [8] donde se obtienen una significativa mejora utilizando unidades lineales rectificadas (ReLU, Rectified Linear Units) y que han sido comprobadas en [2], (ver Figura 3), donde se muestra una comparación en la tasa de error durante el entrenamiento donde el uso de las unidades ReLU supone que se realice seis veces más rápido que con el uso de las funciones de activación comúnmente utilizadas.

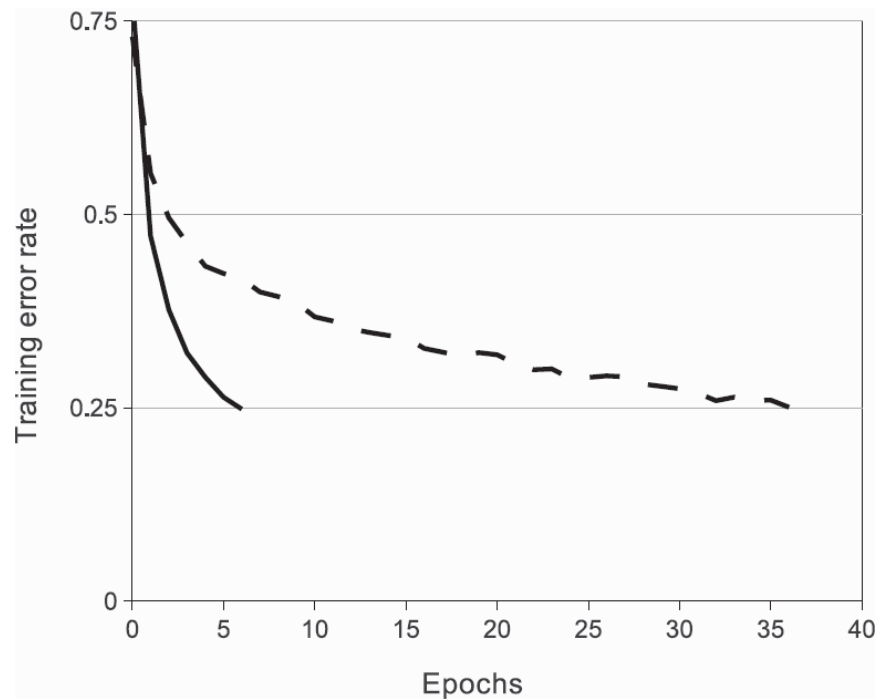


Figura 3. Entrenamiento de red de cuatro capas, con ReLU en la línea continua y con la función tanh en la línea discontinua [2]

La actuación de una capa de neuronas suele ser suficiente para resolver problemas básicos como una clasificación sencilla de datos, siendo necesario el uso de una composición de varias de estas capas para afrontar casos más complejos donde aumentar el número de neuronas supone un mayor gasto computacional en comparación al uso de varias capas, como se ha avanzado previamente. Durante la clasificación las capas se irán adquiriendo las diferentes características de la imagen modificando los pesos que componen los filtros en cada capa durante el entrenamiento, de manera que, una vez entrenada la red se pasará una imagen nunca vista por ella consiguiendo a la salida un mapa de características que nos den información acerca de las formas que componen la imagen; en este trabajo se realizará un entrenamiento que otorgue pesos a los filtros de forma que se disponga un mapa de características similar para clases comunes y suficientemente distintas entre las diferentes clases.

Algoritmo back-propagation

La obtención de los filtros se realiza utilizando un razonamiento de filtrado adaptativo de las imágenes donde los pesos de los diferentes filtros se van modificando en las posteriores iteraciones, en base a una función de error; se realizan dos trayectos a través de la red, el primero para poder predecir la salida que debería acercarse a la clase a la cual pertenece la imagen coincidiendo cuando el reconocimiento es perfecto. Y el segundo para adaptar los pesos en función del error cometido.

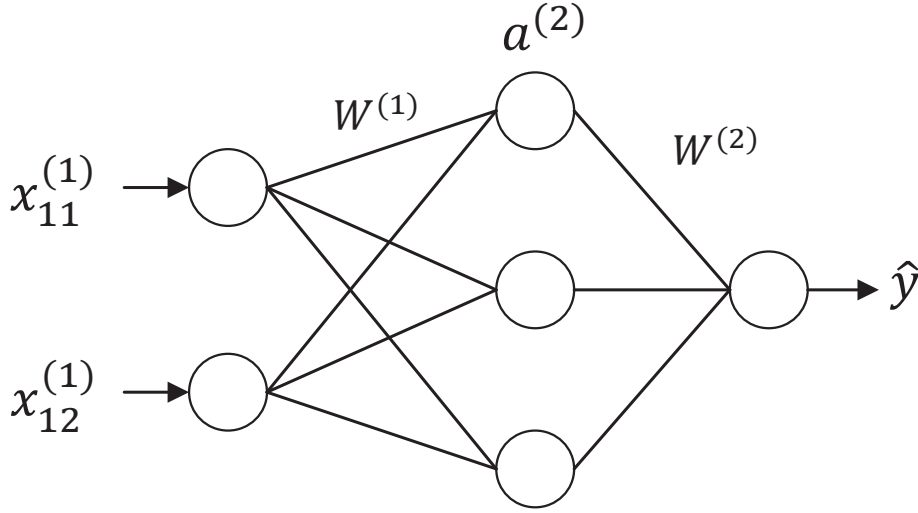


Figura 4. Diagrama básico de red neuronal

El diagrama anterior servirá para entender fácilmente el algoritmo, similar al visto en [9], se tendrán entonces señales de 2 muestras $x_{11}^{(1)}, x_{12}^{(1)}$, expresando entre paréntesis la capa en la que se encuentra o de la que proviene para las señales y los filtros respectivamente.

La primera capa será de entrada, dado que posteriormente tendremos 3 unidades o neuronas en la capa oculta, tendremos 3 pesos $w_{11}^{(1)}, w_{12}^{(1)}, w_{13}^{(1)}$ para el caso de la primera neurona y $w_{21}^{(1)}, w_{22}^{(1)}, w_{23}^{(1)}$ para la segunda neurona de entrada, si además al producto escalar para cada caso los llamamos $z^{(2)}$, se puede llegar a la siguiente relación matricial

$$\begin{bmatrix} x_{11}^{(1)} & x_{12}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} = \begin{bmatrix} z_{11}^{(2)} & z_{12}^{(2)} & z_{13}^{(2)} \end{bmatrix} \quad (2)$$

Pudiendo expresarse entonces como

$$\begin{aligned} x^{(1)} \cdot W^{(1)} &= z^{(2)} \\ a^{(2)} &= f(z^{(2)}) \end{aligned} \quad (3)$$

De la misma forma obtenemos el paso por la siguiente capa, de salida, y generando una predicción de la clase a la que pertenece la señal, \hat{y} , que se acercará a la clase y original.

$$\hat{y} = f(z^{(3)}) = f(a^{(2)} \cdot W^{(2)}) \quad (4)$$

Una vez obtenida una predicción de la clase habiendo realizado un trayecto hacia delante, se realiza una actualización de los filtros dependiendo del error entre la predicción y la etiqueta de la imagen, esta actualización se realizará desde la salida hacia la entrada dando nombre al algoritmo.

La actualización vendrá definida para minimizar el error de predicción de la red; para ello, se definirá la función de coste como el error cuadrático medio entre la predicción de la clase y la clase de la imagen

$$J = E[(y - \hat{y})^2] = \frac{1}{2} \sum (y - \hat{y})^2 \quad (5)$$

La búsqueda de cada peso que minimice la función de coste realizando evaluaciones de ésta, sería una tarea que llevaría mucho tiempo, por lo que se utiliza la derivada de la función de coste de forma que se busca ir reduciendo el error cuadrático medio, por lo que su derivada será negativa, hasta que sea nula, lo que indicará que hemos llegado a un mínimo, conociendo este algoritmo de gradiente descendente ya que si la función de coste es expresada como un vector su derivada consistiría en aplicar el operador gradiente.

Siguiendo el mismo ejemplo anterior y aplicando la regla de la cadena tenemos

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial W^{(2)}} = -(y - \hat{y}) f'(z^{(3)}) \cdot a^{(2)} \quad (6)$$

siendo f' la derivada de la función de activación, pudiendo expresar entonces la derivada de la función de coste como

$$\begin{aligned} \frac{\partial J}{\partial W^{(2)}} &= \delta^{(3)} \cdot a^{(2)} \\ \frac{\partial J}{\partial W^{(2)}} &= a^{(2)T} \cdot \delta^{(3)} \end{aligned} \quad (7)$$

realizando una transposición al vector $a^{(2)}$ para que coincidan las dimensiones cuando tengamos varias señales en el batch de entrenamiento.

Realizando el mismo razonamiento, obtenemos la variación de la función de coste para los pesos $W^{(1)}$

$$\begin{aligned}\frac{\partial J}{\partial W^{(1)}} &= \delta^{(3)} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W^{(1)}} = x^{(1)T} \cdot \delta^{(3)} \cdot W^{(2)T} \cdot f'(z^{(2)}) \\ \frac{\partial J}{\partial W^{(1)}} &= x^{(1)T} \cdot \delta^{(2)}\end{aligned}\tag{8}$$

Siguiendo la explicación anterior, podemos extender el algoritmo al número de capas ocultas entre la de entrada y la de salida, que se tengan en el sistema y de las unidades de cada una; la principal característica de este proceso es aprovechar la regla de la cadena para poder calcular las derivadas en función de los resultados de la capa anterior, las más próximas a la de salida, además de la función derivada de la activación y la señal de entrada, es aquí donde gana importancia el uso de la función sigmoide ya que su derivada se puede expresar como producto de la misma función

$$\begin{aligned}\frac{\partial f(x)}{\partial x} &= \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) \cdot \frac{e^{-x} + 1 - 1}{1 + e^{-x}} \\ \frac{\partial f(x)}{\partial x} &= f(x) \cdot [1 - f(x)]\end{aligned}\tag{9}$$

Para que en el sistema, los filtros, se ajusten a los modelos no lineales que poseen las diferentes clases, harán falta varias iteraciones buscando que la variación de los filtros nos lleve a una convergencia la cual indicaría que la red neuronal está entrenada dependiendo del error cuadrático que obtenemos.

La actualización de los pesos se lleva a cabo realizando un algoritmo adaptativo, a partir de un factor de innovación, la derivada de la función de coste, y de versiones anteriores de los filtros; la velocidad con la que se realiza la búsqueda del mínimo del error cuadrático medio gracias al gradiente se verá reflejado en la tasa de aprendizaje, η , de forma que cuanto mayor sea más rápidamente se avanzará en el aprendizaje, su elección suele ser de forma empírica y en este trabajo se guiará por los utilizados en otros trabajos siendo $\eta = 0.001$ [1].

Por otra parte se definirá un parámetro de inercia denominado momentum, μ , y que, multiplicando al filtro anterior, marcará la fiabilidad que se tienen de las versiones anteriores de los filtros.

Por tanto obtenemos la siguiente expresión de la actualización de los filtros

$$\widehat{W}^{(i)}(n+1) = \mu \widehat{W}^{(i)}(n) + \eta \frac{\partial J}{\partial W^{(i)}(n)} \quad (10)$$

Una vez alcanzada la convergencia de los pesos de los filtros, al realizar el paso de una imagen por las diferentes capas de la red entrenada, se pueden obtener una imagen que recoge la información del mapa de características que llevarán a la activación de una u otras unidades en la capa de salida para poder clasificar la imagen en una de las clases que existen.

Arquitectura utilizada

Uno de los aspectos a tener en cuenta del sistema es el número de capas ocultas que tendrá la red de forma que el resultado obtenido dé un resultado satisfactorio pero el utilizar un número excesivo de capas puede generar un aumento en el número de cálculos que no permitan que sea practicable el uso de esta técnica.

La arquitectura que seguirá la red neuronal será similar a las seguidas en los artículos [2] y [1] con una tasa de error 16.4 % y 14.8%, aunque variarán las imágenes que se utilizarán el entrenamiento de la red neuronal, ya que utilizan la base de datos de Imagenet con imágenes etiquetadas con objetos que se quieren detectar en ellas.

El tamaño de los filtros será otro punto a tener en cuenta, si los filtros son demasiados grandes, la información relevante recogida en cada imagen se dispersaría espacialmente mezclando aspectos importantes y diferenciadores entre las clases.

Siguiendo a [1] partiendo de la arquitectura de [2] y mejorando sus resultados, la red se compone de dos partes, la primera destinada a la obtención del mapa de características y otra para clasificar la imagen en función las formas que tenga la imagen.

La parte de adquisición de las características, se crea a partir de la capa de entrada y se usan cuatro capas para obtener el mapa de características, cada capa contiene subcapas.

Tipos de subcapas

La arquitectura de la red neuronal convolucional se compondrá por la concatenación de un grupo de capas que a su vez estarán formadas por un grupo de subcapas que realizarán diferentes tareas para la extracción del mapa de características, a continuación se explicará el cometido de cada una de las subcapas y sus expresiones matemáticas siguiendo la notación del manual del software Matconvnet [10], observando que se denomina i la fila i -ésima, j a la columna j -ésima y d a la componente d -ésima de su profundidad, su tercera dimensión.

De la misma forma H se refiere al número de filas de la imagen, W al número de columnas y D a su profundidad utilizando una comilla cuando se refiere a una selección parcial de éstas.

- Subcapa convolucional: este tipo de subcapa será parte esencial de la red neuronal que irá variando durante el entrenamiento, es la que contiene los pesos de los filtros que se adaptarán para extraer las características de las imágenes.

$$y_{i''j''d''} = b_{d''} + \sum_{i'=1}^{H'} \sum_{j'=1}^{W'} \sum_{d'=1}^D f_{i'j'd'} \cdot x_{i''+i',j''+j'-1,d',d''} \quad (11)$$

Siendo f los pesos de los filtros y b un factor de bias asociado a cada filtro $H \times W$.

- Subcapa unidad rectificada linealmente (ReLU): esta subcapa realizará la función de activación vista en la explicación teórica detrás del algoritmo, en este caso la función $\max(x, 0)$.

$$y_{ijd} = \max(x_{ijd}, 0) \quad (12)$$

- Subcapa de normalización: se realiza una normalización espacial de un pixel entre canales o de las salidas de las unidades implementando un funcionamiento similar al de las neuronas reales favoreciendo a los píxeles o unidades adyacentes de mayor actividad e inhibiendo a los otros.

$$y_{ijk} = x_{ijk} \left(\kappa + \alpha \sum_{t \in G(k)} x_{ijt}^2 \right)^{-\beta}, \text{ con } G(k) \subset \{1, 2, \dots, D\} \quad (13)$$

- Subcapa pooling: se aplica el filtro de pooling centrado en subáreas de la imagen con sus centros separado 2 píxeles y con tamaño de 3 píxeles para el filtro, calculando la media de los píxeles en el que se centra y sus adyacentes en función

del tamaño del filtro y su paso produciendo así un pooling solapado haciendo que se produzca una mejora de entre 1.4% y 1.2% [11], [12], siendo utilizado su alternativa, el modo maxpooling que en lugar de utilizar su media utiliza el máximo del conjunto de píxeles

$$y_{i''j''d} = \max_{1 \leq i' \leq H', 1 \leq j' \leq W'} (x_{i''+i'-1, j''+j'-1, d}) \quad (13)$$

- Subcapa dropout: se aplica la técnica llamada dropout, que se basa en seleccionar y desactivar unidades en la salida de la subcapa anterior con probabilidad 0.5, de forma que la salida de las unidades seleccionadas no contribuirán en el algoritmo hacia adelante ni hacia atrás, de esta forma evitamos su co-adaptación, o en otras palabras, que una unidad no requiera de la activación de otras para poder dar un resultado correcto ya que en cada iteración del algoritmo las imágenes del entrenamiento pasarán por una arquitectura diferente, dando robustez al entrenamiento.
- Subcapa softmaxloss: aplica dos funciones a la vez en la misma subcapa, por una parte la función softmax utilizada en clasificación de varias clases cambiando el rango de un vector a valores entre cero y uno, y la función log-loss que calcula la entropía cruzada entre la distribución de probabilidad de las predicciones y las etiquetas reales; se obtiene una probabilidad para cada clase, eligiendo posteriormente la de mayor probabilidad.

$$y = - \sum_{ij} \left(x_{ijc} - \log \left(\sum_{d=1}^D e^{x_{ijd}} \right) \right) \quad (14)$$

Cada una de estas subcapas se interconectarán formando así la estructura de la red neuronal artificial, siguiendo a [1], se tendrá una red con 8 capas que se colocan de la siguiente manera:

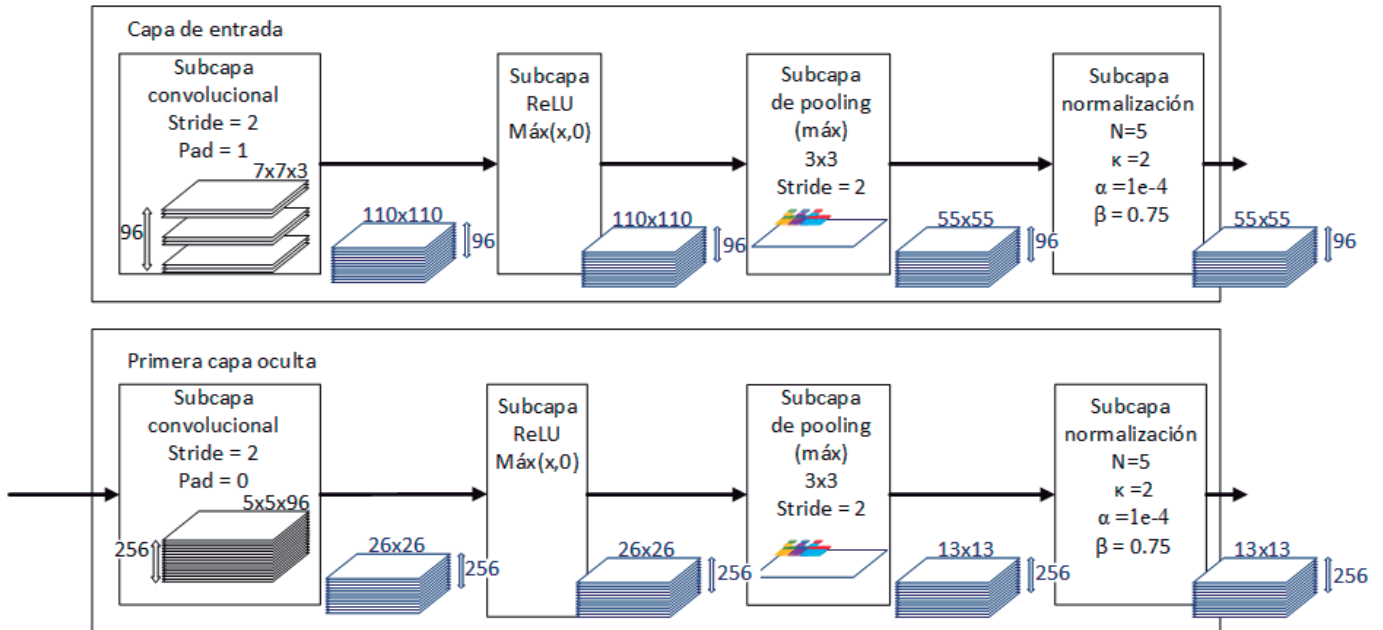


Figura 5. Capa de entrada y primera capa oculta de la red neuronal, los cuadrados negros representan los filtros y los azules las imágenes.

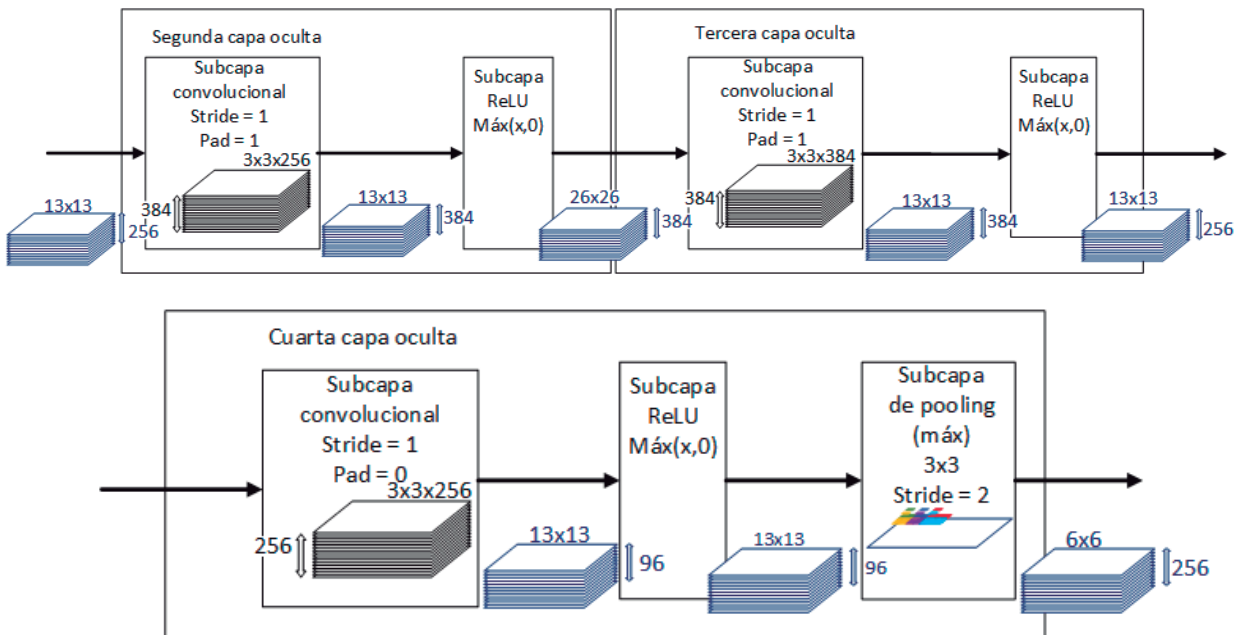


Figura 6. Segunda, tercera y cuarta capa oculta.

Donde se puede observar como las 5 primeras capas, la de entrada y las 4 siguientes, realizarán la extracción del mapa de características de las imágenes que pasarán posteriormente por concatenación de las capas restantes para realizar la clasificación como se muestra en la Figura 7 con un diagrama más simplificado.

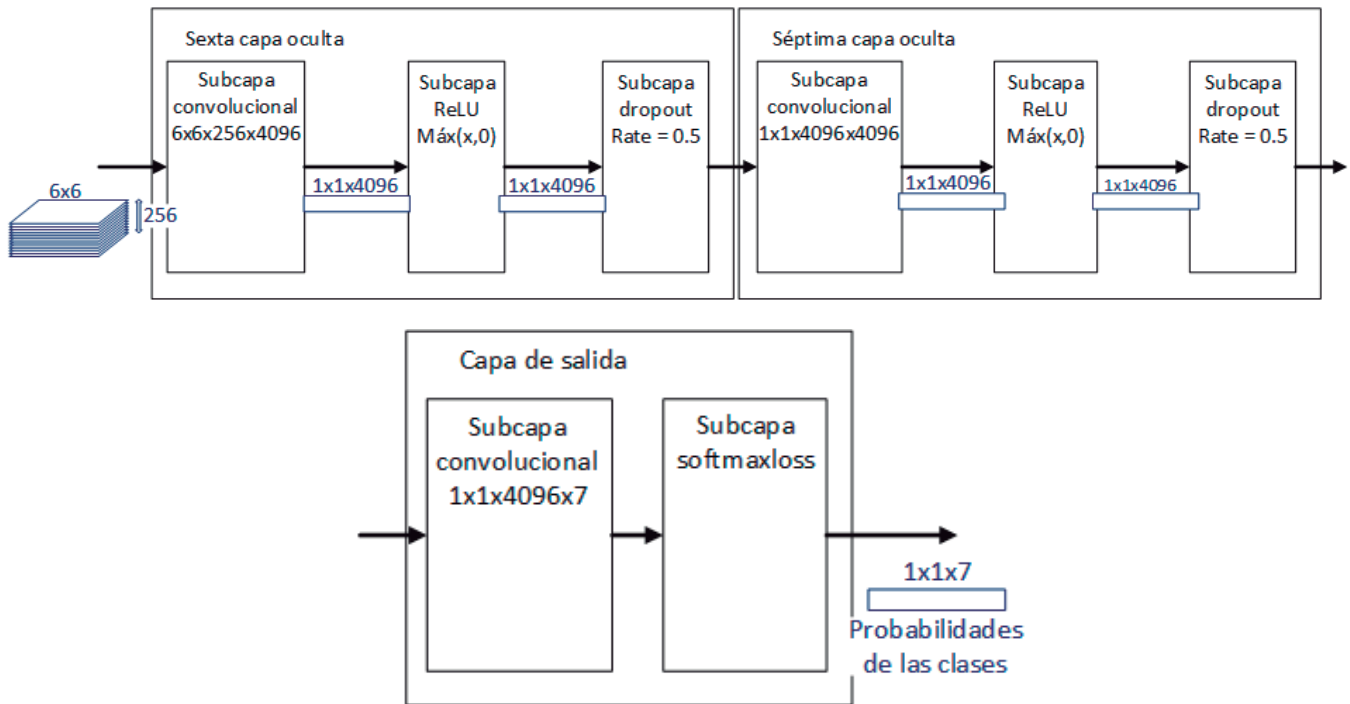


Figura 7. Fase de clasificación, sexta, séptima capa y de salida.

En la parte de extracción de las características vemos como la primera capa consta de una capa convolucional donde tendremos la capa de entrada con los primeros filtros adaptados a las formas que se quieren detectar dentro de la escena; a la salida de ésta, tendremos una subcapa de unidades ReLU, realizando una primera aproximación del mapa de características, después se realiza una normalización de la salida para evitar problemas de saturación en los siguientes estados, cuando realice el filtrado y sume los canales, con una subcapa de normalización y un pooling.

Tras esta primera capa se pasará su salida a través de las otras cuatro capas restante que se corresponden a una sucesión de subcapas de convolución y unidades ReLU exceptuando la segunda y quinta capa donde se volverá a realizar un pooling a la salida de su ReLU correspondiente; en este punto, una vez entrenado el sistema, debería obtenerse el mapa de características propio de cada clase para las distintas imágenes.

Esta arquitectura sigue un patrón descendente con respecto a la dimensión de los filtros ya que a medida que se avanza a través de la red, los filtros se van haciendo cada vez más pequeños de forma que coincidan con la matriz de salida de la subcapa de pooling reduciendo su tamaño $\frac{L_i - (L_f - S)}{S}$ de lado, siendo L_i una de los lados de la imagen (son imágenes cuadradas), L_f el tamaño de uno de los lados de los filtros (también cuadrados), y S la distancia entre los píxeles en que se centra el filtro de pooling.

La fase de clasificación se realiza con dos capas compuestas por una subcapa convolucional configurada de forma que los filtros en ella tienen el mismo tamaño que las salidas de las capas anteriores, de esta forma realizará un producto escalar,

multiplicando y sumando cada componente del mapa de características con las componentes del filtro que contendrá los pesos que se adaptarán para poder realizar la clasificación de la clase a la que pertenece la escena, por tanto además se tendrá una subcapa ReLU a la salida de la subcapa de convolución que realizará a activación de las diferentes unidades en función de la clase a la que pertenezca.

En la última capa, conocida como capa de salida, tendremos la última subcapa de convolución la cual tendrá tantos filtros como clases se tengan, y por último una subcapa softmaxloss, la cual se encargará de calcular el error de predicción que ha tenido la red en función de la etiqueta que tiene asociada la imagen para poder realizar el algoritmo backpropagation y modificar los filtros para las siguientes iteraciones.

Cálculos en GPU

Para llevar a cabo los cálculos necesarios para este TFG se va a realizar una transformación sobre los datos, trabajando con ellos de forma matricial, algo que para el procesamiento de imagen es mucho más intuitivo gracias a la equivalencia en forma de las imágenes como matriz, introduciendo así la idea de convolución como la realización del producto de la entrada con los pesos como un filtrado de la imagen con una matriz que contenga dichos filtros. Es aquí donde se han producido grandes avances debido a la mejora del hardware utilizando tarjetas gráficas actuales muy optimizadas para el procesamiento de imagen, incluso en 4K, paralelizando los cálculos y uniendo los resultados.

En este trabajo se utilizarán los recursos del clúster Hermes, de la Universidad de Zaragoza, más concretamente con una tarjeta gráfica nVIDIA Tesla M2090 de 6Gb de memoria RAM, con una versión de CUDA 5.5 y una capability de 2.2, esto último permite a MATLAB poder realizar las operaciones con los recursos hardware que ofrece la tarjeta gráfica y poder explotar su usabilidad.

El uso de este método suele ser, en casos de procesamiento de imagen, el de detectar la presencia de uno o varios objetos, mientras que el realizado en este trabajo es el de reconocer la escena de la imagen a partir de las características, presentes en la misma, extraídas por los filtrados a través de las capas de la red.

Posteriormente se reflejará la mejora que supone realizar el entrenamiento con una tarjeta de vídeo en lugar de hacerlo por CPU.

Implementación y herramientas

La implementación de una red neuronal convolucional puede llevarse a cabo mediante muchas opciones siempre teniendo en cuenta que se necesita el hardware pertinente para poder ejecutar las diferentes librerías existentes que tienen diferentes variantes según el sistema operativo que se utilice o el lenguaje en el que se lleve el trabajo.

Uno de los lenguajes más utilizados es Python, ejecutable en Windows y GNU/Linux, ofrece ventajas como la simplicidad y flexibilidad [13] que resultan más fácil la implementación y manejo de las herramientas que ofrece como su ejecución en tiempo real, una de las librerías que se utilizan con este lenguaje es Theano que proporcionan funciones implementadas para poder utilizar los recursos que ofrece las tarjetas gráficas.

Otro es C++, utilizado también para poder realizar aplicaciones en tiempo real, se ejecutan en sistemas operativos de Windows, utilizando la librería cuda-convnet, nos ofrece, al igual que el resto de opciones, la posibilidad de poder implementar el entrenamiento y ejecución de la red neuronal convolucional en dicho lenguaje aprovechando al máximo posible la tarjeta vídeo.

Por otra parte, tenemos la posibilidad de poder llevar a cabo la creación y entrenamiento de la red neuronal en MATLAB, escogida en este trabajo fin de grado, como es el uso de la librería VLFeat que junto a un toolbox llamado MatConvNet, se pueden ejecutar implementaciones de las funciones básica que son necesarias explicadas anteriormente; aunque no se puede utilizar para diseñar soluciones en tiempo real, explota la capacidad de MATLAB para realizar operaciones con imágenes siendo tratadas como matrices, siendo además un aliciente el haberlo utilizado durante el grado teniendo una mayor desenvoltura que con el resto de lenguajes.

Desarrollo del trabajo

Una vez establecidas las bases en las que se sustentará este trabajo, la primera fase consistiría en obtener un conjunto de datos, lo suficientemente grande para poder obtener una correcta convergencia en el aprendizaje de la red neuronal.

Base de datos

Inicialmente se realizó una búsqueda de diferentes bases de datos de las imágenes, datasets, que pudiesen ayudar en los resultados, pero la mayoría se componían por un gran número de clases. Siendo la más utilizada la base de datos denominada Imagenet con 1000 clases, la cual posee muchas imágenes resultado de tener muchas clases, pero no las suficientes cuando se tenían pocas clases como es este caso.

Por tanto se decidió crear un programa que fuese capaz de generar la base de datos automáticamente ya que el etiquetado de todas las imágenes es un trabajo arduo que en condiciones normales se realiza por grupos extensos de personas que realizan esta actividad manualmente ya sea a la hora de la creación de la fotografía o etiquetando imágenes ya creadas.

Para poder crear esta aplicación se realizó una pequeña búsqueda de las diferentes posibilidades que se tenían a partir de una búsquedas automatizadas en Internet de las imágenes necesarias, por tanto se estudió la posibilidad de utilizar la interfaz de programación de aplicaciones, API, que ofrece Google, una de las empresas con mayor trayectoria en este sentido, pero visto que para llevarlo a cabo se necesitaban conocimientos de otros lenguajes que no se habían adquirido se buscó una alternativa con otro buscador de otras de las empresas líderes en el sector de la tecnología como es Bing, perteneciente a Microsoft. La cual aunque no disponía de una API que poder utilizar, permitía la posibilidad de poder realizar peticiones a través de una conexión mediante una URL la cual tendría que tener el mismo formato que generan los buscadores automáticamente.

Una vez observado que se podía realizar una conexión fácilmente se eligió crear el programa en Java que facilitaba la conexión a la URL sin necesidad de utilizar librerías complementarias como es el caso de C++ por ejemplo; así pues, la base del funcionamiento del programa era crear la URL pertinente para cada clase, una vez creada se realizaban peticiones al buscador y se guardaban los códigos fuentes que

devolvía, ya que es como se comunican con los navegadores, donde contenían las URLs en donde residían las imágenes resultado de la búsqueda.

Una vez obtenidos los ficheros fuente de la página, bastaba con buscar las cabeceras que diferenciaban las URLs de las imágenes de la búsqueda con otros objetos más relacionados con la presentación de la página web; obtenidas las direcciones bastaba con realizar la descarga de dichas imágenes paralelamente haciéndola más eficiente y tratando los posibles errores que podían ocurrir, como se explica en el Anexo 1 donde se detalla el funcionamiento del programa.

Descargadas las imágenes, se necesitaba tener ordenadas cada una de las clases así como datos que pudiesen ser relevantes para poder diferenciarlas como el tamaño, o la dirección de donde se ha descargado para una posible utilización de forma remota; en este aspecto se utilizó la librería SQLite de código abierto, la cual permite crear bases de datos relacionadas en diferentes lenguajes utilizando C++ ya que aún se estudiaba la posibilidad de utilizar cuda-convnet mencionado en el apartado anterior; a través de una organización jerarquizada de las carpetas contenedoras de las imágenes, el programa recorría las diferentes imágenes comprobando que se pueden abrir gracias a otra librería, OpenCV, y posteriormente almacenando sus datos, ruta, dimensiones, etc. dicho funcionamiento se detalla en el Anexo 2 de este trabajo.

En este punto, se había optado por utilizar la opción de MATLAB para llevar a cabo el proyecto, gracias a que SQLite era compatible y muy fácil de usar con MATLAB no supuso ningún esfuerzo importante el poder utilizar la base de datos ya creada que disponía de 29.539 imágenes repartidas en las 7 diferentes clases.

Dado que aun así era un número pequeño de imágenes, siguiendo la línea de [2] y [1] se le aplicó al conjunto de imágenes una técnica denominada data augmentation.

Data augmentation

Este método de obtención de imágenes nos permite aumentar artificialmente el número de imágenes y, por consiguiente, reducir el sobreajuste de la red neuronal como resultado de disponer de pocas imágenes, ya que se generan cambios en las imágenes originales de las clases de forma que sean parecidas pero que aporten información diferente de una misma imagen.

Se basa en un pre procesamiento de las imágenes que se explican a continuación:

- **Escalado:** dado a la cantidad de imágenes recolectadas hasta ahora, es inevitable que las propiedades del archivo no sean iguales en todas, en esta apartado se aborda el hecho de que existan imágenes de diferentes ratios de aspecto. Ya que la capa de entrada de la red neuronal necesita unas dimensiones concretas de

imágenes cuadradas, es necesario realizar un reescalado de la misma pero si no tenemos en cuenta el ratio de aspecto de la imagen, si se modifican las filas y columnas de forma convencional, producirá una deformación en los elementos que componen las escena por lo que el entrenamiento de la imagen no será adecuado ya que encontraría formas que no corresponden modificando los pesos para adquirir dicha forma en clases a las que no corresponde. Por tanto se realiza una selección cuadrada de la imagen aprovechándola lo máximo posible.

- **Rotación:** ya que en un caso de aplicación real del sistema la imagen podría recogerse girada a diferencia a las recogidas en Internet, en general destinadas a que la habitación en concreto se vea bien, por lo que darle esta robustez al sistema es importante. Dado que el cambio de coordenadas en los píxeles que produce la rotación de la imagen, genera zonas nulas en zonas donde no había presencia de imagen y ahora forman parte de la nueva imagen se ha creado un algoritmo de interpolación del cuadrado que aprovecha la imagen girada partiendo del punto central de la imagen para excluir los ceros creados por la rotación que durante el entrenamiento se entenderían como algo normal en la habitación en concreto, al final se obtiene una imagen rotada y escalada de la imagen original.
- **Recorte aleatorio:** se recogen zonas aleatorias dentro de la imagen que contengan el 75% de la imagen original, lo que correspondería a la realización de una imagen en diferentes posiciones de la habitación.
- **Reflexión:** para obtener más fotos que no sean iguales a las anteriores, se realiza una reflexión con respecto al eje horizontal de las imágenes anteriores que ayudarán a que cambie la distribución de las formas dentro de la clase.

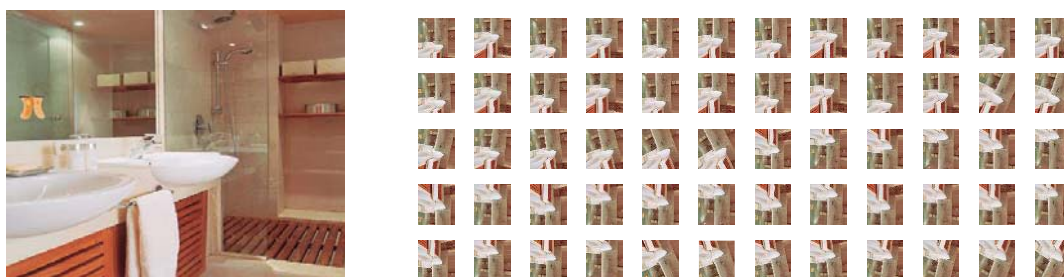


Figura 8. Data augmentation aplicado a una de las imágenes

Realizando el escalado; 8 rotaciones desde -28° a 28° sin tener en cuenta la rotación de 0° , que corresponde a tenerla original; recortes de la imagen obteniendo 10 cortes de cada imagen, y reflejando todas ellas horizontalmente, obtenemos alrededor de unas

40 imágenes por cada imagen original, valor que dependerá del ratio de aspecto de cada imagen de forma que habrá más cuanto mayor sea la diferencia de filas, en el ejemplo de la Figura 8 es de 60 imágenes.

Se obtienen 1.767.346 imágenes en total, aunque, debido a que no se tienen el mismo número de imágenes para todas las clases, la que contenga menos imágenes en total marcará el número máximo de imágenes por clase, con lo que se tiene un total de 642.124 imágenes en total (91.732 por clase).

Programación de la red neuronal

Para la programación de la red neuronal, es necesario entender cómo se implementan las funciones que permitirán realizar el entrenamiento, para ello se va a seguir un pequeño ejemplo de entrenamiento de una pequeña red para reconocer caracteres.

La red se compone de una capa de entrada formada por una subcapa convolucional y una de pooling; otras dos de la misma forma que conseguirán el mapa de características que servirá para la clasificación de los caracteres mediante la siguiente capa full-conectada junto a una subcapa ReLU y la capa de salida con otra subcapa convolucional con una softmaxloss.

El objetivo será ver como realiza el aprendizaje la red neuronal en alto nivel para prever cual será el comportamiento que tendrá nuestra red neuronal durante el entrenamiento y corroborar que es correcto. Este ejemplo, forma parte de un conjunto de ejercicios obtenidos de [14] y que propone Andrea Vevaldi junto a Andrew Zisserman. Siendo el primero uno de los programadores que implementaron el código para entrenar una red neuronal y que ha sido aprovechado para entender las funciones implementadas, creando funciones necesarias para la inicialización de la red neuronal y el acceso a las imágenes ya que las estructuras de la base de datos eran diferentes a la Imagenet usada en el código ejemplo. Eso se explicará con mayor detalle en el Anexo 3.

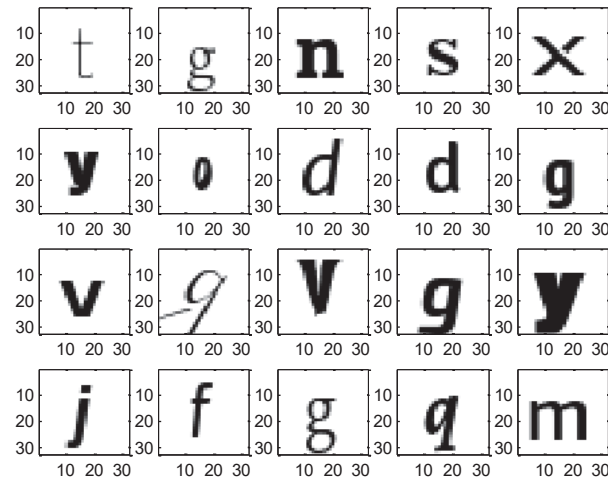


Figura 9. Representación imágenes escogidas aleatoriamente de la base de datos.

En la siguiente imagen podemos ver como los pesos de la primera capa de la red neuronal han tomado forma tras converger a una solución aceptable; al filtrar las imágenes con estos filtros, el resultado es una imagen donde destacan unas formas por encima de otras, como pueden ser circunferencias por ejemplo.

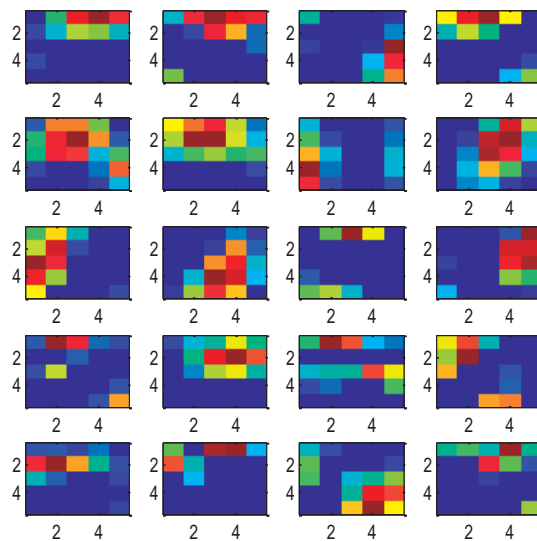


Figura 10. Representación de los filtros de la capa de entrada de la red.

El entrenamiento se realiza con un ratio de aprendizaje $\eta = 10^{-3}$, y según podemos ver en la Figura 11 la convergencia se produce cerca de la décima iteración, a partir de la cual la variación del porcentaje del error se mantiene constante con un error mínimo.

Dado que el error durante la fase de entrenamiento, que se calcula con las imágenes destinadas para esa labor, es cada vez menor debido a que el sistema se ajusta a dichas imágenes no puede ser un condicionante final de convergencia sino un aspecto de control del comportamiento de la red. Una intuición correcta del funcionamiento de la red neuronal, sería observar los resultados que obtienen las imágenes en la fase de

validación, donde se utilizan imágenes que no participan en la adaptación de los pesos de las diferentes capas.

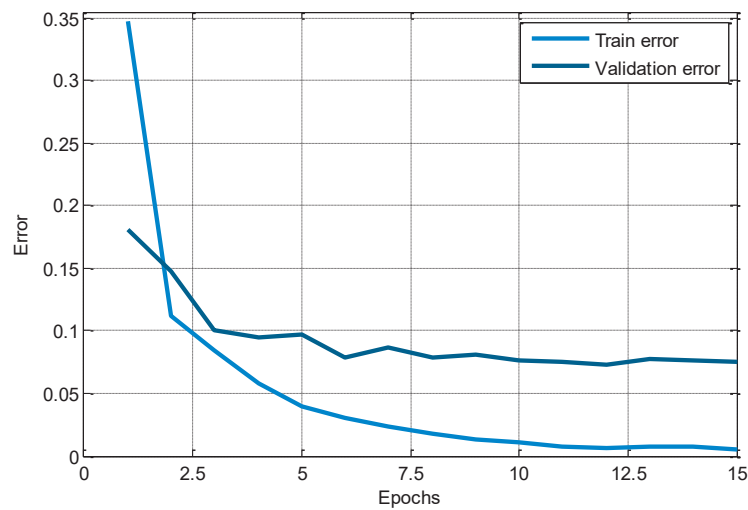


Figura 11. Evolución del porcentaje de error en entrenamiento y en validación.

Podemos ver además que el error cuadrático medio va disminuyendo durante el paso de las iteraciones en el entrenamiento. Una vez que se mantiene estable, con una baja variación (0.1 en el error cuadrático medio), es una medida de la convergencia que puede alcanzar la red neuronal, similar al porcentaje de error, nos da indicaciones de cómo se aproxima al mínimo de la función de coste que, en general, no varía demasiado en esos casos.

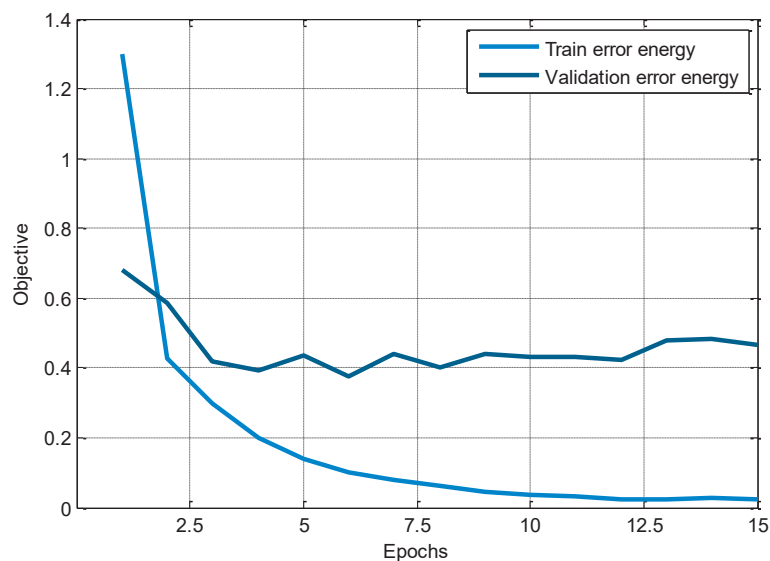


Figura 12. Evolución del error cuadrático medio.

Se puede definir que unos resultados similares pueden llevarnos a considerar que la red neuronal ha convergido y que obtiene unos buenos resultados. Ya que el resultado obtenido ha sido de alrededor de un 7.5% de error en validación, a la vez que vemos que el coste ha dejado de descender, por lo que concluimos que el sistema ha encontrado el mínimo.

Este tipo de entrenamiento, en general, suele orientarse a minimizar el error en lugar a favorecer la velocidad de convergencia del entrenamiento ya que al necesitarse un gran número de iteraciones, es más prudente aprovechar ese tiempo de entrenamiento en llegar a una solución adecuada que ofrecer bajas prestaciones de forma rápida.

Desde un punto de vista del comportamiento del sistema, podemos observar el efecto de las diferentes capas en una imagen, en la Figura 13 podemos ver como los filtros se van ajustando a las formas que posee la imagen

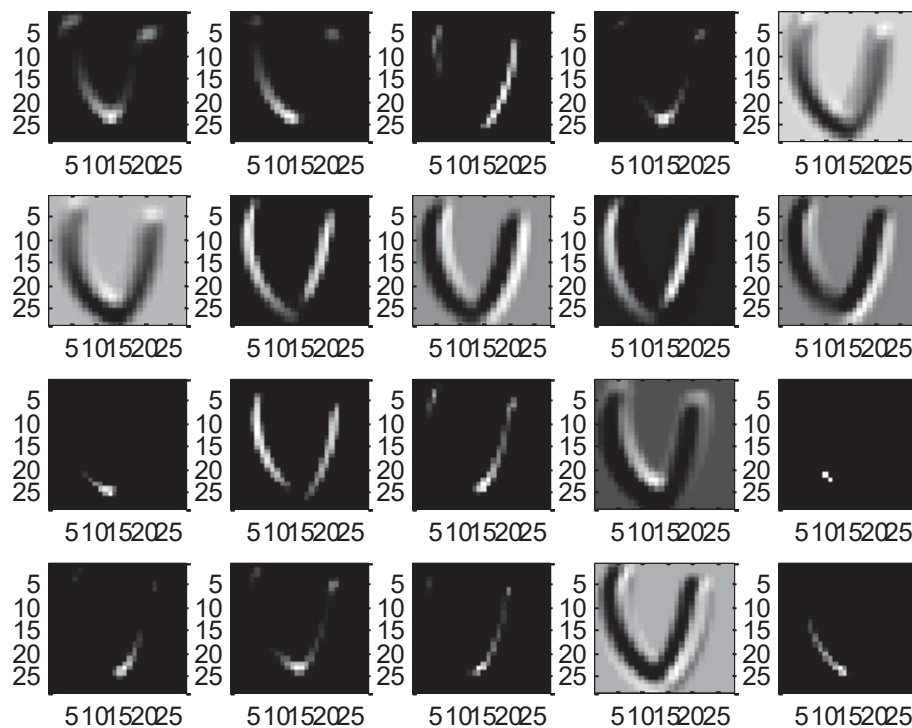


Figura 13. Salida de los filtros en la capa de entrada para la letra v.

A través de las siguientes capas, se realizarán diferentes filtrados para poder obtener el mapa de características que definirán la clase de la imagen, hasta llegar a la subcapa de ReLU, donde se tendrán las activaciones de las diferentes unidades en función de la clase a la que pertenezca la imagen

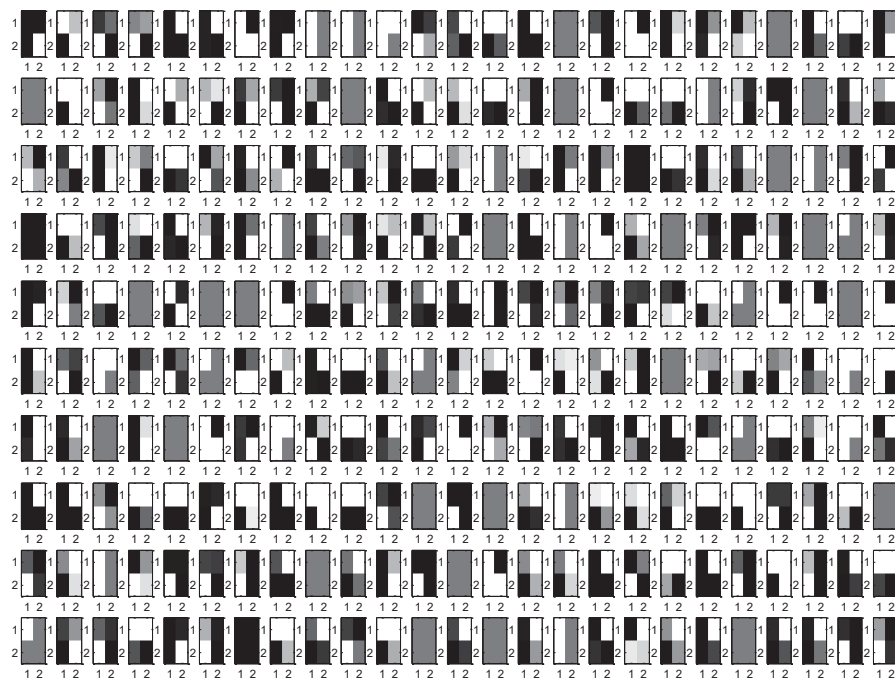


Figura 14. Salida de la subcapa ReLU, activación de las unidades.

La salida de la última capa será un vector con la probabilidad para cada carácter del alfabeto, recogido en la base de datos, como podemos ver en Figura 15, el sistema ha errado en la predicción siendo el carácter más probable una u, siendo el carácter real la letra v, el ejemplo ha sido seleccionado para que le sea difícil acertar al sistema, observando que es normal el error, ya que la formas que presenta la letra, sobretodo la curva en la parte inferior, podría llevar a equívoco.

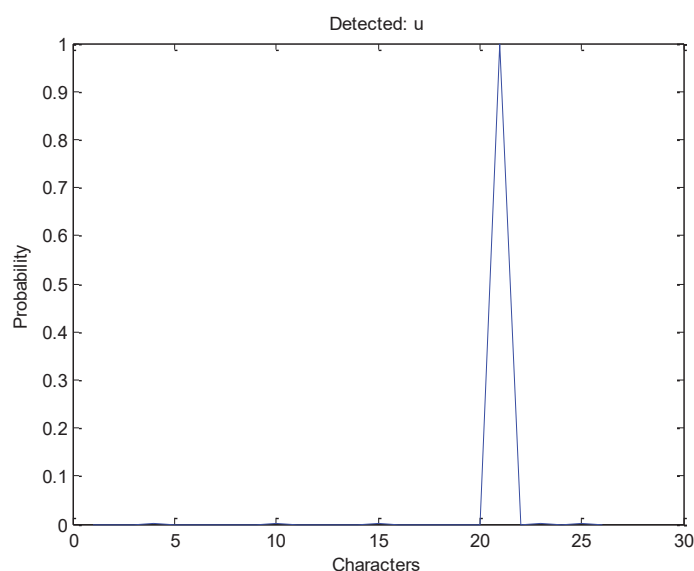


Figura 15. Predicción del carácter.

Una vez que se entendieron las limitaciones y características de las redes neuronales, se pasó a realizar varias pruebas previas antes del entrenamiento final.

Pruebas de entrenamiento y posibles problemas

Para la creación de la red neuronal se crearon funciones en MATLAB para poder iniciar una red siguiendo una estructura específica, en concreto la seguida en [1] y representada en la Figura 5 y Figura 6 donde podemos ver su composición siendo explicado en el Anexo 4 su formato en MATLAB. Se realizaron varias pruebas en condiciones diferentes para observar los posibles errores que puedan ocurrir durante nuestro entrenamiento.

Sustitución de imágenes

Debido a que el gran número de imágenes que se disponen en la base de datos, es inviable cargar previamente todas las imágenes en una matriz de MATLAB para poder acceder a ella fácilmente, por tanto se realiza un guardado de la ruta donde reside la imagen para poder cargar únicamente las imágenes que sean necesarias.

En caso de que no se pueda acceder a la imagen, se puede sustituir por otra de forma que pueda ser solucionado el problema de acceso para la siguiente iteración.

No es recomendable sustituirla por una imagen que sea la media de la base de datos ya que al sustraerla al principio del proceso daría salidas muy pequeñas o nulas, interfiriendo así en el cálculo del gradiente de la función de coste en la actualización de los filtros.

Otra opción sería crear una imagen artificial aleatoria dentro del rango dinámico de una imagen convencional, entre 0 y 255, de esta forma no produciría un cambio tan brusco en la energía del error durante el entrenamiento y evitaría que el entrenamiento se interrumpa en caso de fallo.

En el caso más extremo, podría suceder que el número de imágenes aleatorias a la que se accede sea demasiado grande, entonces dado que también hay presentes imágenes correctas de la base de datos, la red intenta adaptar los pesos a las imágenes correctas, pero consideraría que las otras clases a las que no puede acceder corresponde a una clase que intenta predecir.

Si observamos la Figura 16, podemos ver como la convergencia es muy rápida con respecto a los casos comunes donde esa tasa de error sería más propia de un elevado número de iteraciones de entrenamiento, por tanto ya podemos intuir que no está siendo correcto el método en el que se está realizando el aprendizaje.

En la Figura 17 se muestran los resultados obtenidos hasta la séptima iteración y vemos como hay filtros que intentan adaptarse a la clase que se ha dejado para realizar una predicción adecuada; pero sin embargo, la existencia de aleatoriedad en las imágenes artificiales creadas cambia constantemente los otros filtros reduciendo la fiabilidad del entrenamiento.

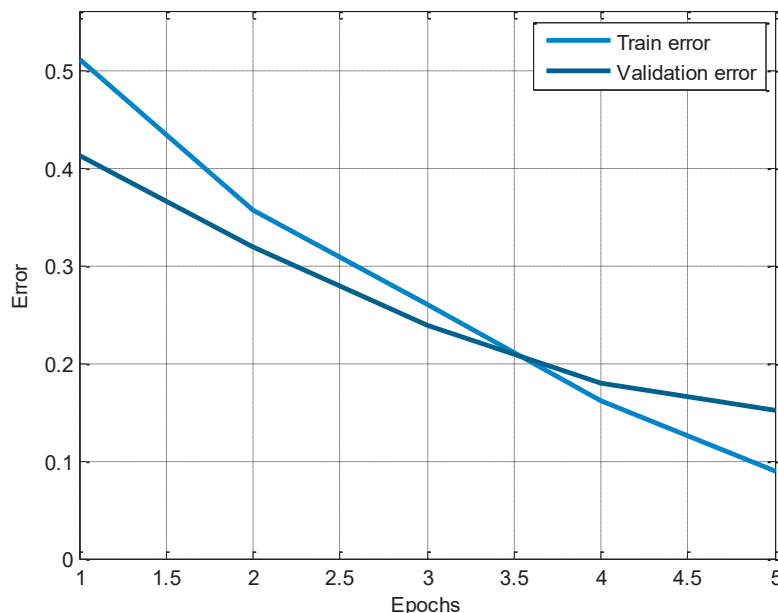


Figura 16. Evolución de porcentaje de error para un conjunto de muchas imágenes artificiales aleatorias

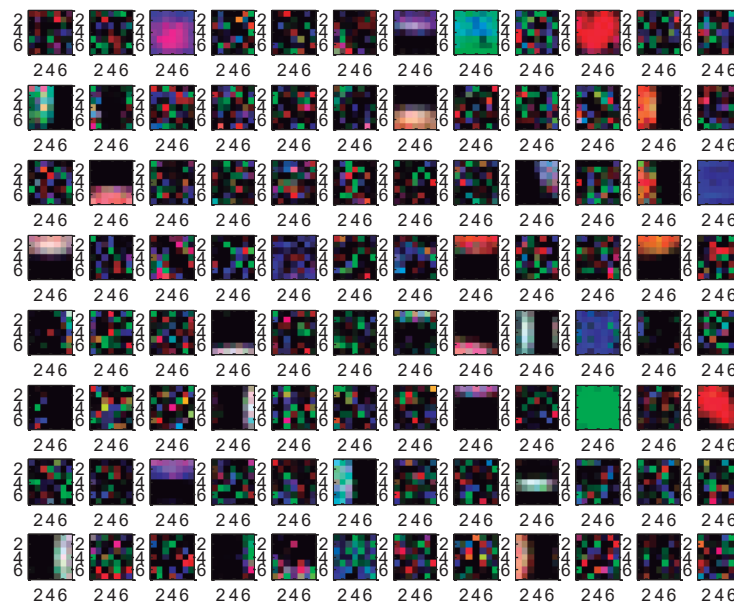


Figura 17. Convergencia de los filtros de la capa de entrada para varias entradas de imágenes aleatorias

Esto se debe a que en realidad, asigna las imágenes artificiales a una de las clases que no puede acceder a las imágenes, eligiendo siempre esa clase y contando como error al resto cuando la entrada es de tipo aleatorio mientras que si la imagen pertenece a

alguna de las clases, encontrará las características coexistentes en la clase que ha entrenado la red y la clase a la que pertenece la imagen, activando las unidades y dando un falso positivo; sea como sea el resultado es totalmente erróneo aunque el resultado de la gráfica podría llevarnos a confusión.

El efecto de las imágenes aleatorias sobre los pesos entrenados de la clase correcta afecta de manera poco significativa, si llevamos el proceso a cantidades de imágenes muy altas, del orden de este trabajo, vemos como la presencia de formas concretas, existentes tanto en las imágenes artificiales como en las reales, realizarán una pequeña variación en los pesos de los filtros con respecto a los otros ya que la red pensará que los pesos son adecuados para esas formas en concreto.

Sustracción de la media de la base de datos

Uno de los aspectos críticos en el uso de redes neuronales convolucionales es controlar la entrada y salida de los datos, en este caso imágenes, de forma que el funcionamiento se encuentre dentro de unos rangos acotados.

Se realiza algo similar a una tipificación de las imágenes de entrada que centra el nivel de media cercano al cero de forma que las imágenes de entrada tienen una distribución normalizada.

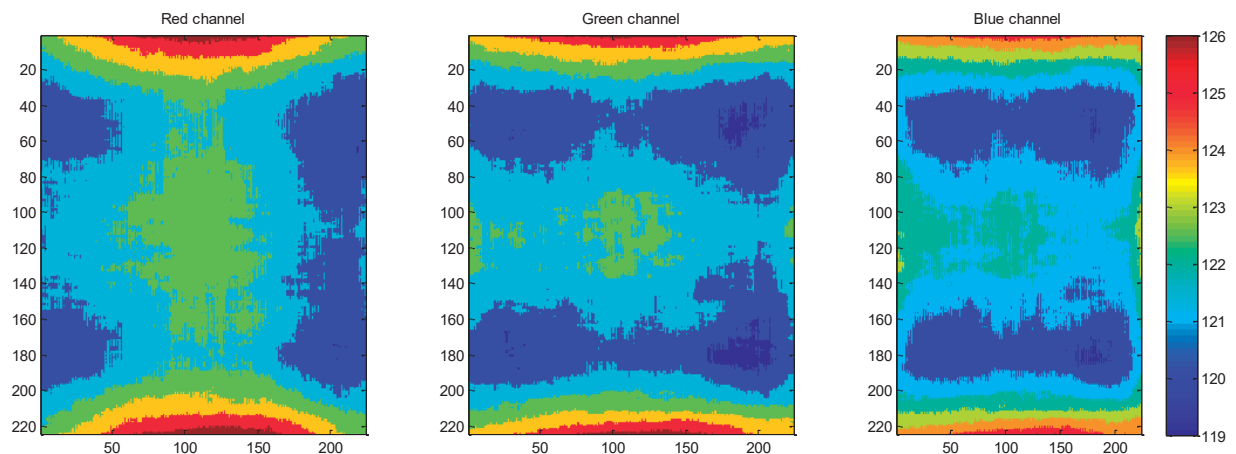


Figura 18. Imágenes medias de la base de datos para cada canal.

El no restar la media de la base de datos no es necesario si el rango dinámico de las imágenes ya se ajusta a niveles entre -128 a 128, donde la media es cero, sin embargo si se utiliza todos los niveles convencionales de las imágenes, puede resultar perjudicial para el proceso con resultados desastrosos.

Si no se resta la media, los datos de entrada serán de niveles altos, lo que corresponderán a que las salidas de las convoluciones con los filtros de las capas ocultas serán altas también, por lo que durante las actualizaciones de los pesos se tratarán con

operaciones de matrices de valores muy altos a partir de las cuales pueden aparecer indeterminaciones en los pesos de los filtros anulándolos completamente y expandiendo su actividad al resto de filtros, en la Figura 19 podemos ver varias iteraciones en las que no se han restado la media a las imágenes, los filtros completamente negros corresponden a los filtros con indeterminaciones en sus componentes.

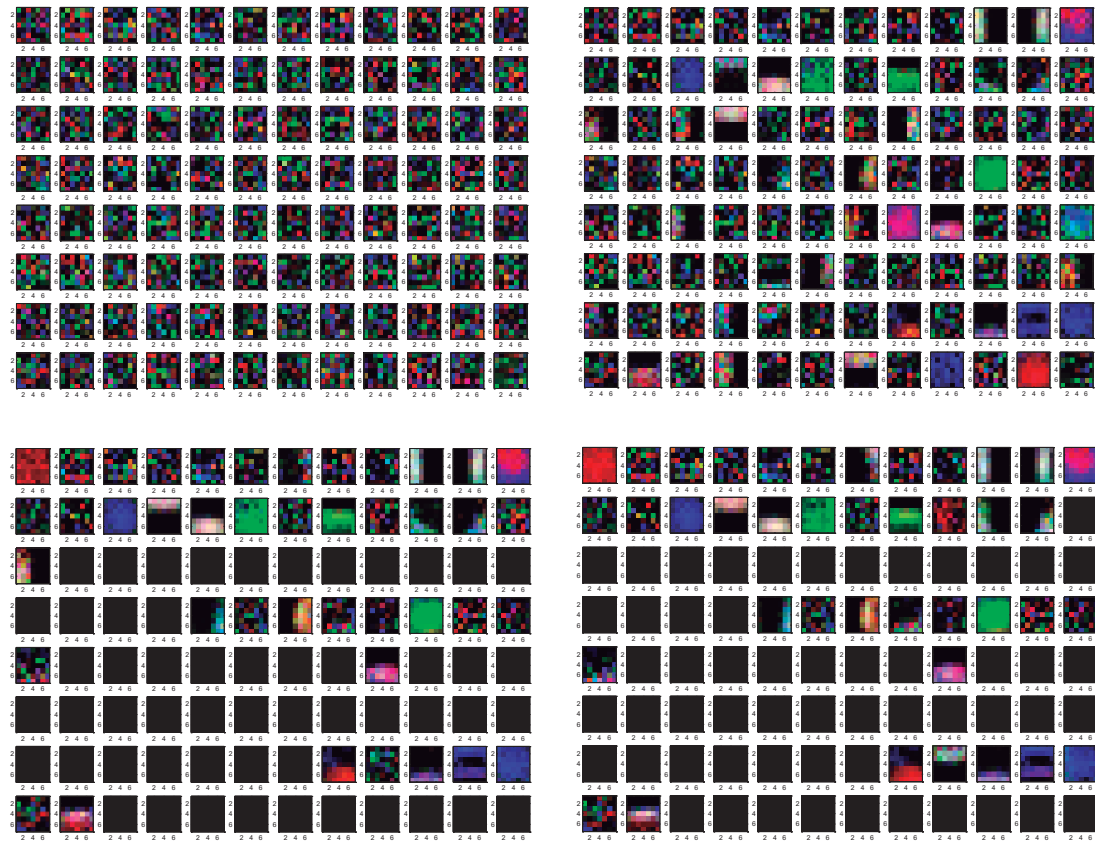


Figura 19. Representación de la iniciación del filtro y de las 3 primeras iteraciones.

En la Figura 20 se puede ver como la inactividad de los filtros producen una salida nula que junto a las salidas no nulas de las convoluciones de filtros se consigue una predicción totalmente errónea, si se deja continuar el entrenamiento, el sistema intenta realizar la predicción aunque, como se puede apreciar, la probabilidad de error se estabiliza en la probabilidad de fallar 6 de 7 intentos para las 7 clases diferentes que existen

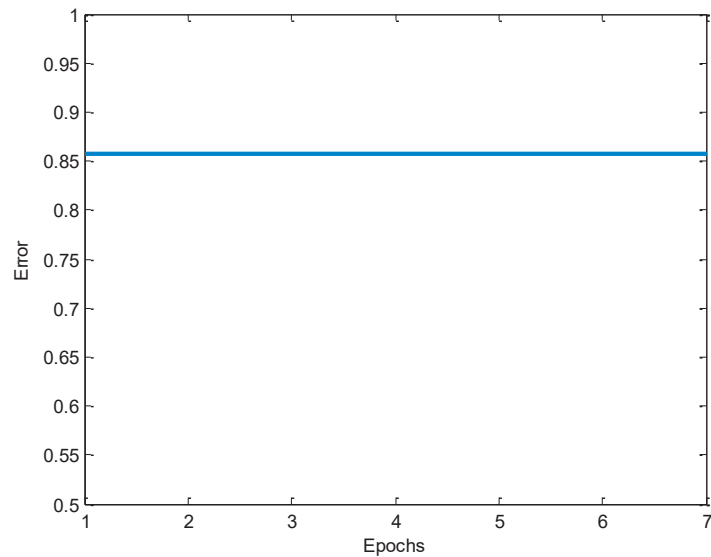


Figura 20. Evolución de la probabilidad de error de una red no funcional

Cabe mencionar que se podría intuir una posible solución reduciendo el factor de aprendizaje, η , de forma que el crecimiento del error cuadrático medio se vea reducido por este factor, pero en realidad se obtiene el mismo comportamiento con mayor lentitud ya que a medida que va progresando en el entrenamiento, llegaría un momento en el que crecería rápidamente el error cuadrático hasta provocar las indeterminaciones que se mencionaban anteriormente.

Número de imágenes

Este trabajo se ha basado en conseguir un elevado número de imágenes, en caso de tener pocas imágenes no se conseguiría obtener información suficiente para que pueda alcanzar la convergencia.

Ante un número escaso de imágenes podría llegarse a pensar la posibilidad de poder llegar antes a los pesos que nos den una solución más rápidamente utilizando, nuevamente, el ratio de aprendizaje.

En la Figura 21 se comprueba el efecto de realizar esta iniciativa

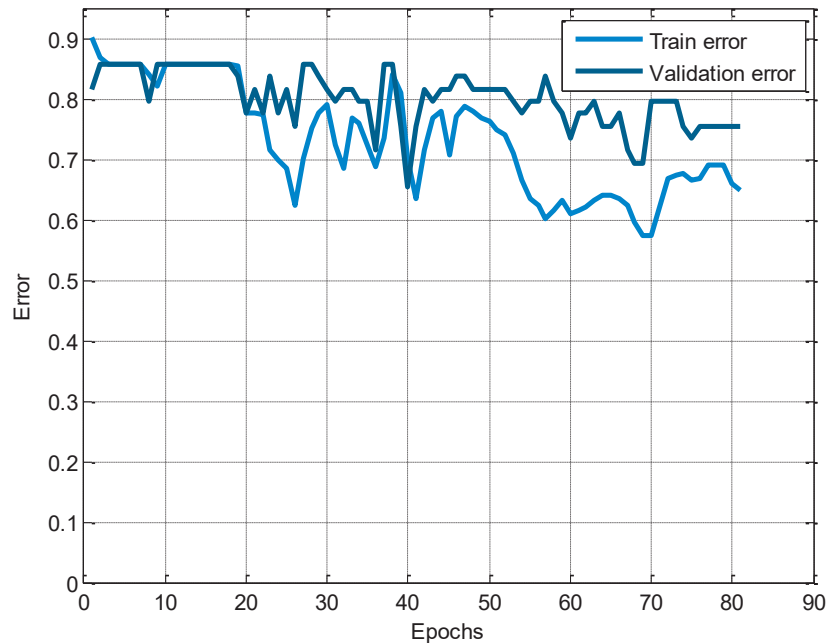


Figura 21. Evolución de la probabilidad de error con un conjunto muy pequeño de 1000 imágenes.

Vemos como es totalmente inútil el modificar η , ya que se puede apreciar la aparición de variación aleatoria del porcentaje del error, perdiendo totalmente el control del funcionamiento de la red neuronal.

Aquí hay que realizar una consideración respecto al número de imágenes, tan perjudicial es tener pocas imágenes como un número desbalanceado de imágenes entre clases, como es de esperar, si una clase posee más imágenes que otras, la red neuronal modificaría los valores de tal forma que daría más prioridad a la clase dominante pasando por alto la información aportada por las otras clases.

Entrenamiento de la red neuronal convolucional y resultados

Se ha llevado a cabo el entrenamiento de la red neuronal creada, mostrando un comportamiento no esperado en un problema de esta índole.

Mientras se realizaba el entrenamiento se observó, tras unas pocas iteraciones, que el sistema había alcanzado rápidamente unos valores bajos de probabilidad de error, en únicamente 5 iteraciones.

Se utilizó manualmente la función creadora de la red neuronal para comprobar su correcta composición, al ver que se formaba correctamente con los parámetros

propuestos en [1], se pasó a verificar que el número de imágenes sea el mismo para todas las clases verificando que así es en el histograma de la Figura 22.

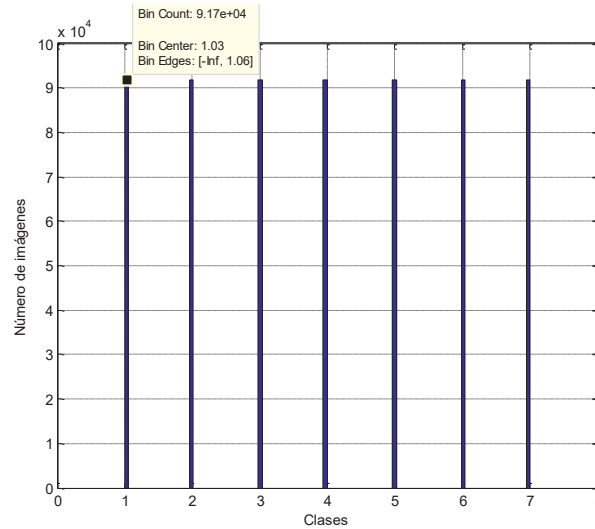


Figura 22. Distribución por clases de las imágenes de la base de datos.

Visto que las clases se distribuían equitativamente, se realizó una comprobación de la asignación de las imágenes a entrenamiento y validación, aquí se había cometido un error ya que se había asignado el 85% del total a entrenamiento cuando había que seleccionar el 85% de cada clase como se representa en la Figura 23.

Lo que nos llevaba a que el sistema estaba siendo sobrentrenado ya que únicamente se validaban los resultados de una clase siendo todas las demás imágenes pertenecientes a la actualización de los pesos.

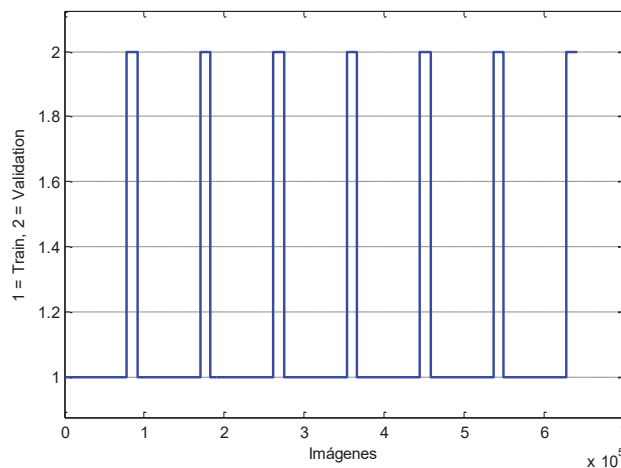


Figura 23. Asignación de las imágenes a entrenamiento o validación.

Una vez resuelto, se veía que se obtenían tasas de error similares, lo que llevó a comprobar los datos obtenidos viendo que tenía un error del 93.51%, lo que lo descartó completamente.

Se revisó el código ya escrito y obtenido del toolbox que se puede encontrar en [5] para encontrar posible fallos o incompatibilidades con la inclusión del código creado, al no encontrar los errores se ha utilizado una estructura diferente a la seguida por [1] junto a la aparición de problemas como los explicados en el apartado anterior se ha utilizado una red similar en cuanto a la composición de las capas ocultas, pero difiere en el tamaño de los filtros que se tienen.

Además no se hace uso del vector de bias, el cual tiene en sus componentes tantos valores como filtros tenga la capa, que suma su valor como continua a la convolución de forma que ayude a la convergencia del filtro, en caso de no utilizarlo, en filtros que tengan un nivel de continua tendrán que ser cada uno de los pesos los que se ajusten a ese nivel.

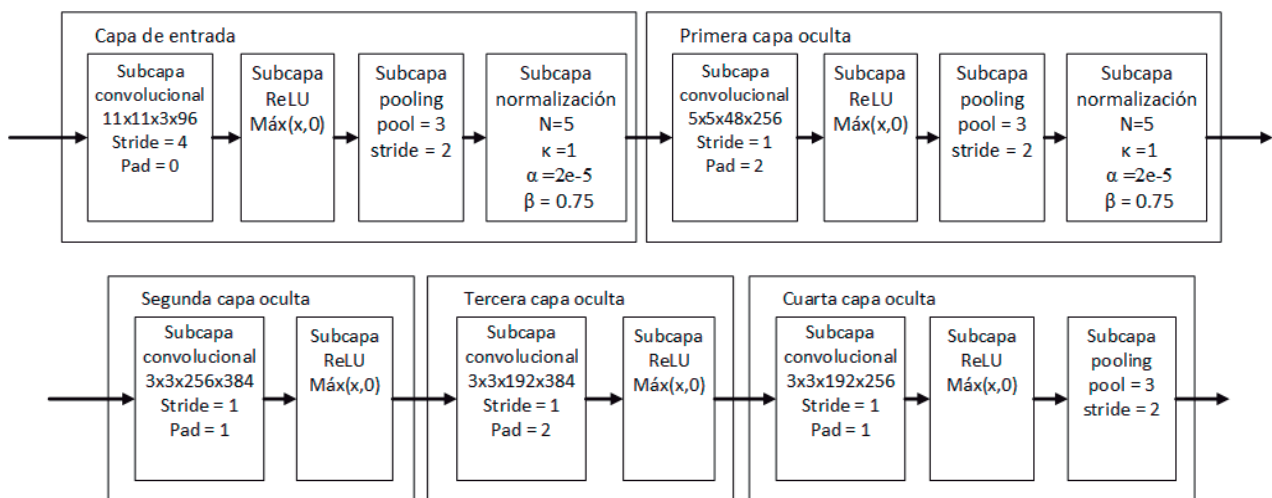


Figura 24. Fase de extracción de características.

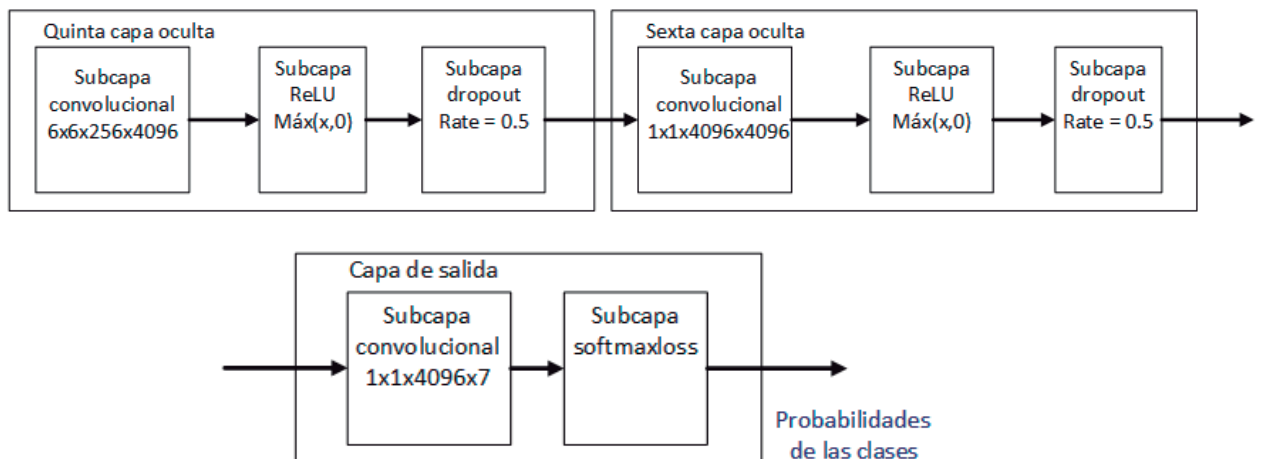


Figura 25. Fase de clasificación.

En este contexto se realiza el aprendizaje con la base de datos de 642.124 imágenes a las que ya le han sido aplicadas el método de data augmentation seleccionando aleatoriamente entre los conjuntos que tengan más imágenes, se utilizarán 545.804

imágenes para el entrenamiento de los pesos y 96.320 para la validación del entrenamiento que nos ayudará determinar si la red es lo suficientemente adecuada con un tamaño del lote de 128 imágenes suficientemente pequeño con respecto a la base de datos para dar independencia al entrenamiento de las imágenes predecesoras de la que está llevando a cabo la actualización de los pesos, en la Figura 26 podemos ver un ejemplo de las imágenes que componen la base de datos.

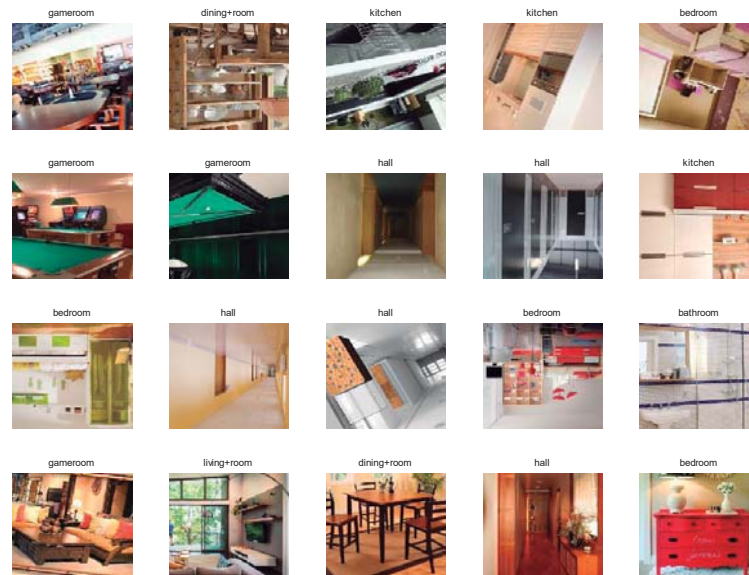


Figura 26. Representación de 20 imágenes aleatorias pertenecientes a la base de datos.

Se realiza el entrenamiento con una tarjeta gráfica perteneciente al clúster Hermes mencionado en el apartado de cálculos en GPU, como comparación se ha realizado previamente un intento de entrenamiento en un ordenador de sobremesa facilitado por el equipo investigación de visión por computador, CVLab, de la Universidad de Zaragoza.

Consta de una tarjeta gráfica nVidia GeForce 8400 GS, de 512 MB de memoria física y una capability de 1.1 con MATLAB, lo que significa que no es posible utilizar completamente los recursos de la tarjeta gráfica, concretamente la creación de vectores en la GPU.

Con el ordenador de sobremesa funcionando con CPU es inviable realizarlo ya que aun dejando de lado el hecho de que se quedó sin memoria suficiente para albergar los datos intermedios que se generan durante el entrenamiento. Se realizaba el procesado de un lote de 200 imágenes en 1700 segundos, lo que nos llevaría a un tiempo de procesado de una iteración de entrenamiento de 2730 lotes, de los que constaría la base de datos, en cerca de 1289 horas lo que es equivalente a 53.7 días que multiplicado por las 32 iteraciones que se han realizado en este trabajo conllevaría 1718.67 días o más de 4 años y medio (4.7). Lo que marca la importancia del uso de una tarjeta gráfica, que en nuestro caso ha realizado el entrenamiento en un tiempo de alrededor de 7 horas por iteración, poco más de 9 días de tiempo efectivo en realizar las 32 iteraciones.

El resultado del entrenamiento se puede comprobar en la Figura 27 donde se ha decidido que la convergencia es correcta, aunque en realidad es variante, cuando está alrededor del 10% de error en los datos de validación, tanto la probabilidad de error como el error cuadrático medio disminuyen rápidamente en las primeras iteraciones y después lo hace suavemente hasta que sea casi inapreciable.

Por otra parte se puede ver como el tener una base de datos con una gran cantidad de imágenes nos llevan a que en la primera iteración el error ya es bastante bajo con respecto a lo que podría suceder en casos de similar complejidad con menos imágenes.

De la convergencia del error cuadrático medio podemos sacar un factor de coste mínimo, similar a los que se suelen tener en cuenta en métodos de filtrado adaptativo como LMS, ese factor podría ser un tipo de medida de comparación de redes neuronales que nos ayuden a saber si una tiene un mejor o peor funcionamiento sin necesidad de hacer una medida de probabilidad del error teniendo siempre presente que debería ser la misma base de datos para poder comparar estructuras de redes neuronales, o una misma estructura con diferentes bases de datos.

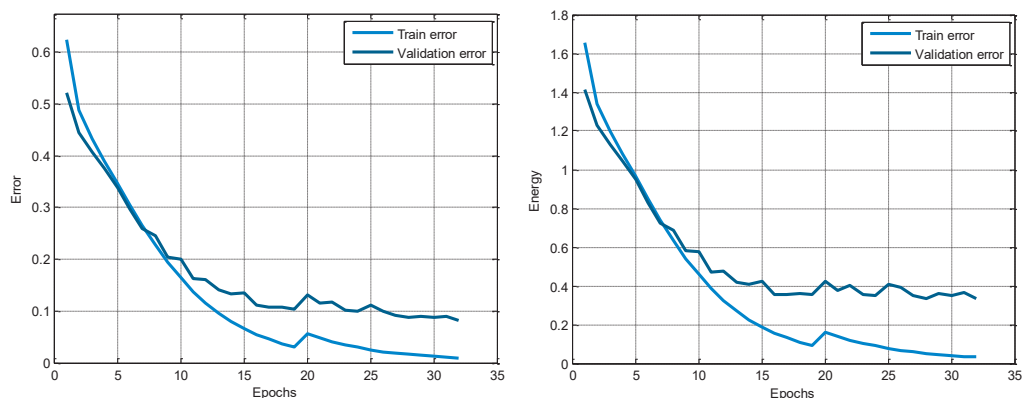


Figura 27. Evolución de la probabilidad de error y del error cuadrático medio en 32 iteraciones

En la Figura 28 podemos ver la forma que han adquirido los filtros tras el entrenamiento, observando que la búsqueda de características se produce tanto en forma como en los colores de la imagen.

Realizando una gráfica de los coeficientes de uno de los filtros de la primera capa, podemos observar que, al igual que en la probabilidad de error y el error cuadrático medio, la variación es más pronunciada en las primeras iteraciones del entrenamiento mientras que va adaptándose a los pesos adecuados.

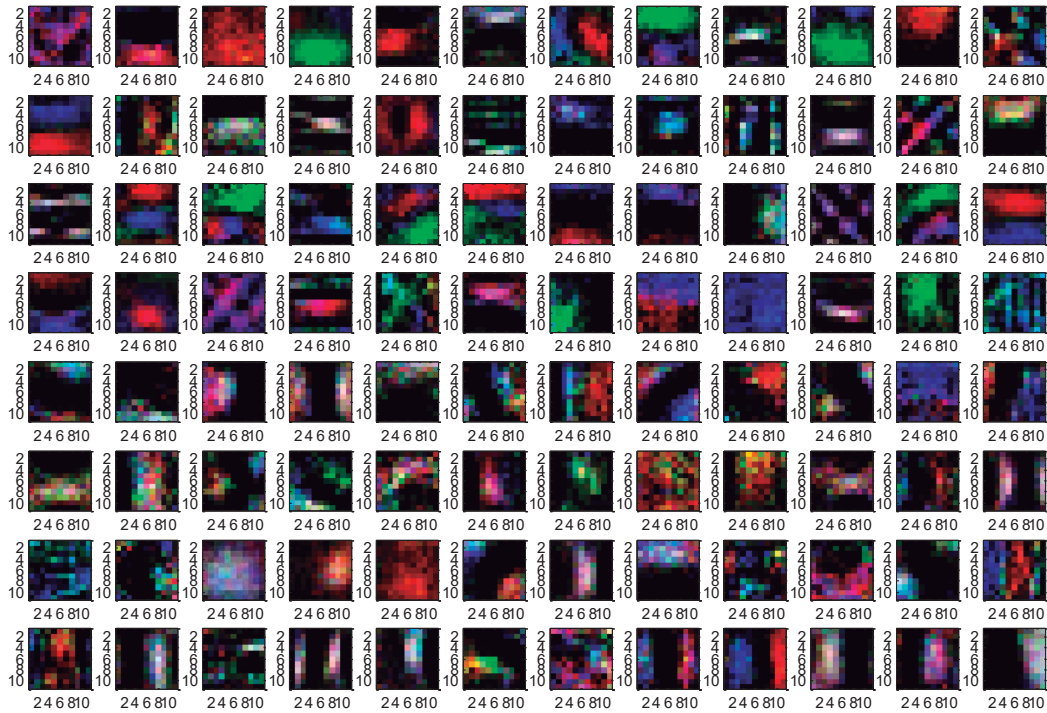


Figura 28. Representación de los filtros de la capa de entrada tras 32 iteraciones de entrenamiento

En la Figura 29 podemos ver la adquisición de las características de una imagen ejemplo con los filtros de la red entrada y la predicción del sistema en la Figura 30

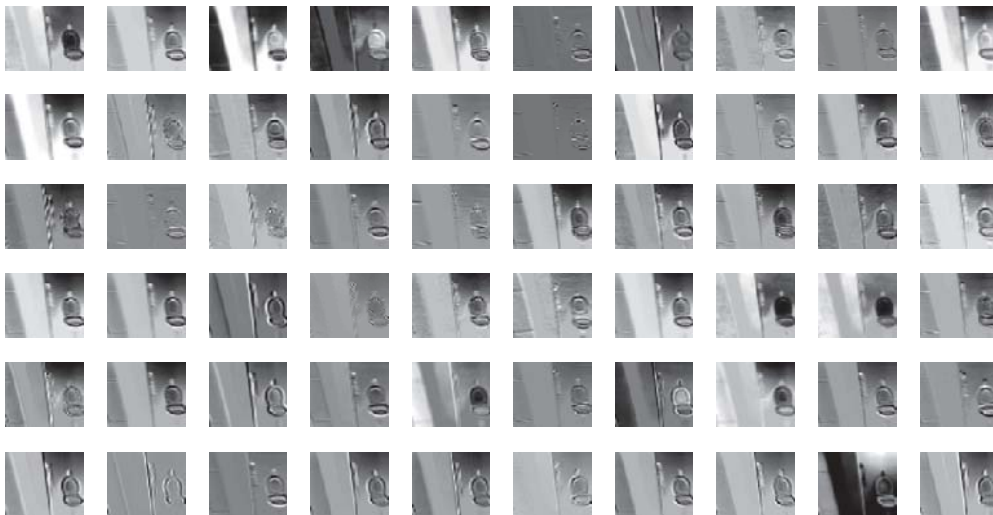


Figura 29. Representación de la salida de 60 filtros de los 96 pertenecientes a la capa de entrada de la imagen de un baño

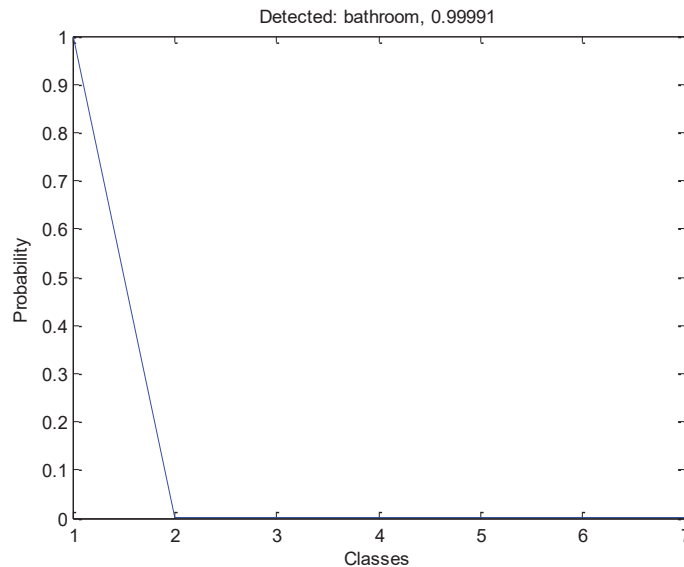


Figura 30. Predicción del sistema para una imagen de un baño una vez entrenada la red neuronal

Para corroborar estos resultados hacen falta un conjunto de datos nunca vistos por la red neuronal, para ello se ha obtenido una pequeña base de datos utilizada para reconocimiento de escenas en interiores [15], de ella se selecciona únicamente las clases para las cuales ha sido entrenada la red, en caso de elegir otras que no pertenezcan, el sistema intentará predecir cuál será la más probable fallando siempre, en caso de que se quisiera dotar a la red de excluir imágenes que no pertenecen a ella se tienen dos opciones o bien establecer un umbral de forma que no se tienen en cuenta los resultados si dicho umbral no es superado, o se podrían añadir imágenes que no pertenezcan a las clases creando una nueva clase que represente la no presencia de las clases que hemos tratado hasta ahora.

Los resultados fueron realmente malos mostrando alrededor de un 90% de error en la predicción de error, revisando el código creado se ha observado un error en la implementación de la entrada de las imágenes a la red; la resta de la media de la base de datos a las imágenes se había realizado con variable enteras sin signo de 8 bits (uint8) comúnmente utilizado en imágenes, por lo que la resta daba un resultado mayor que cero para todas las muestras de la imagen mientras que la forma correcta es realizarlo en tipo de dato single necesario en las funciones implementadas en la librería y que si tienen en cuenta valores negativos.

El resultado de esta desafortunada acción hace que se elimine parte del rango dinámico en imágenes con altas distribuciones entre 0 y la media de la base de datos como se representa en la Figura 31.

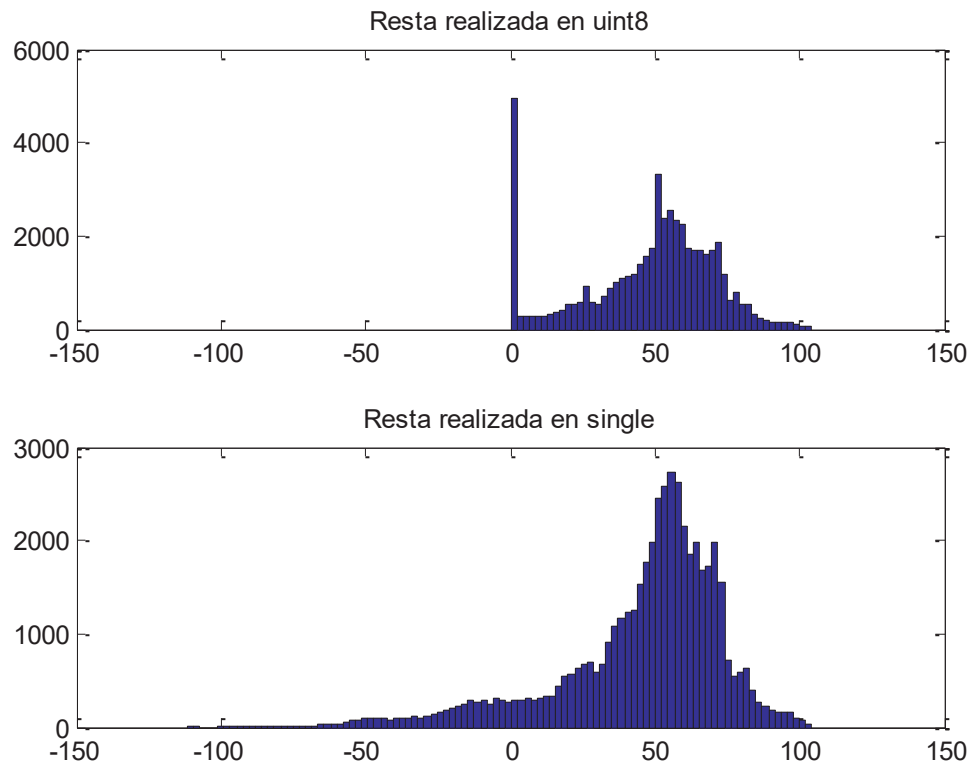


Figura 31. Comparación de distribuciones de las muestras en cada forma de realizar la resta

Debido a que para solucionar este problema conllevaría a realizar nuevamente el entrenamiento de la red neuronal, se ha realizado el estudio de sus prestaciones con la misma base de datos de test pero realizando la resta en uint8, dado que al hacerlo de esta forma se elimina parte de la distribución de la imagen, el reconocimiento se verá entorpecido mostrando la matriz de confusión de la Tabla 2, de donde se desprende que en su conjunto el porcentaje de acierto es menor que el de fallo, exceptuando el caso de los pasillo donde acierta el 70% de las veces, en conjunto, el sistema tiene una probabilidad de acierto de 44.09% debido a la errónea sustracción de la media.

	Bathroom	Bedroom	Dining room	Gameroom	Hall	Kitchen	Living room
Bathroom	49,75%	15,74%	7,61%	4,57%	5,08%	6,60%	10,66%
Bedroom	11,25%	37,69%	10,64%	8,05%	6,99%	4,10%	21,28%
Dining room	3,30%	9,89%	41,39%	10,26%	2,93%	10,62%	21,61%
Gameroom	3,97%	6,35%	27,78%	24,60%	4,76%	15,08%	17,46%
Hall	5,81%	4,65%	6,98%	5,52%	70,06%	3,20%	3,78%
Kitchen	8,17%	8,58%	15,12%	4,63%	3,54%	49,73%	10,22%
Living room	3,40%	12,77%	20,71%	12,91%	4,54%	11,21%	34,47%

Tabla 2. Matriz de confusión las filas representan las etiquetas reales y las columnas las predicciones

Estructura del programa

Este trabajo se ha realizado en partes diferenciadas, como se ha ido explicando durante la memoria, dotando de transparencia cada apartado del proceso con respecto al usuario, de tal forma que es compatible realizar un entrenamiento de una red neuronal con otra base de datos, o se puede crear otra base de datos fácilmente independientemente de cómo es la red neuronal, basta con modificar los campos como el nombre de la clase, los nombres de los archivos, etc.

Programa de descarga de imágenes en Java

La descarga de las imágenes para su posterior almacenamiento se realiza mediante búsquedas automatizadas en Bing, las palabras claves de las búsquedas se almacenarán previamente en un fichero de texto (.txt) con un formato en concreto que diferenciará la clase de la palabra que busca pudiendo entonces unir palabra para aumentar el abanico de búsqueda, además se realiza en varios idiomas (español, inglés, francés e italiano) para aumentar más las posibilidades.

En la Figura 32 se muestra un diagrama de bloques del funcionamiento del mismo, en el Anexo 1 se explicará más detalladamente aspectos del código y solución de errores.

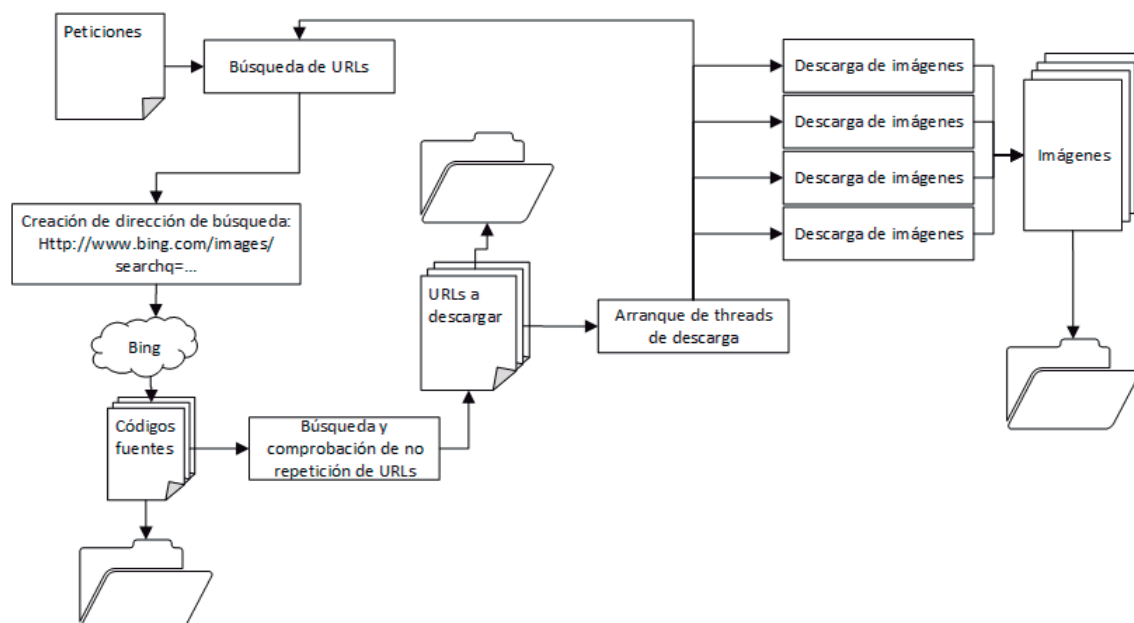


Figura 32. Diagrama de bloques básico del programa de descarga automática en Java.

Programa de etiquetado y construcción de la base de datos en C++

El etiquetado de las imágenes se realizará creando una base de datos de tipo .db, para ello se utiliza la librería SQLite la cual ofrece fácilmente la posibilidad de almacenar texto, números o bloques de bits; se planteó guardar los bits de las imágenes en la base de datos, pero debido al tamaño que ésta alcanzaría era poco eficiente trabajar con ella durante la fase de pruebas de su creación. Se organizará en tres estructuras o tablas que guardarán aspectos diferenciados de las imágenes, éstas se describen a continuación:

- **Imagen:** Guarda los detalles de acceso de la imagen como son la ruta, el tipo de imagen y su tamaño en disco.
- **Datos:** Se almacenan las propiedades de cada imagen como son el nombre de la imagen, la URL de dónde se ha descargado, el formato de la imagen (jpg, png o gif), el número de filas y columnas, la fecha, su etiqueta y si es usada en entrenamiento o no.
- **Clases:** Aquí se encuentran las diferentes clases que se entrenarán en la base de datos siendo éstas los 7 tipos de habitaciones tratados en el trabajo.

En la Figura 34 se puede ver una captura de la base de datos a través de un buscador proporcionado por los creadores de la librería para su gestión de forma gráfica y en la Figura 33 un diagrama de bloques del programa, se explica en el Anexo 2 los detalles de la programación.

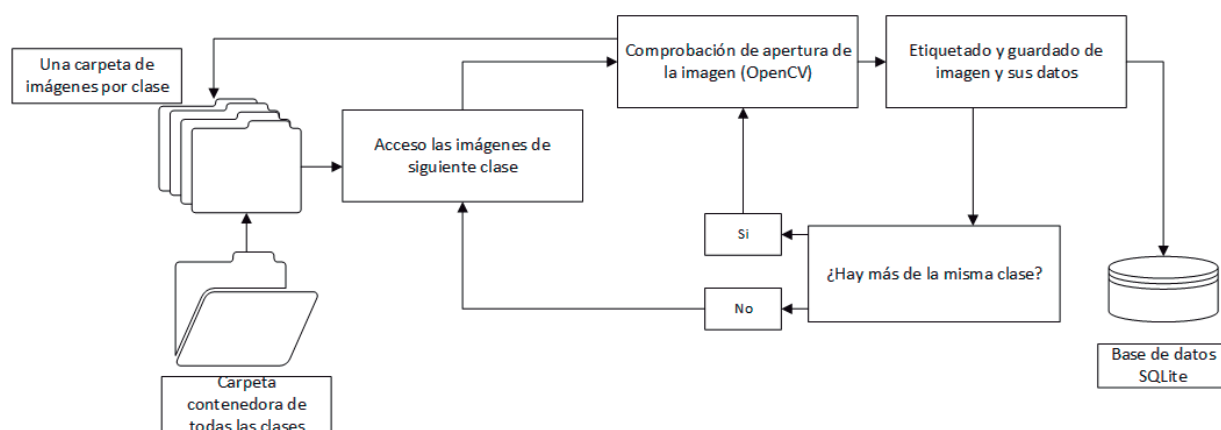


Figura 33. Diagrama de bloques de la creación de la base de datos con C++.

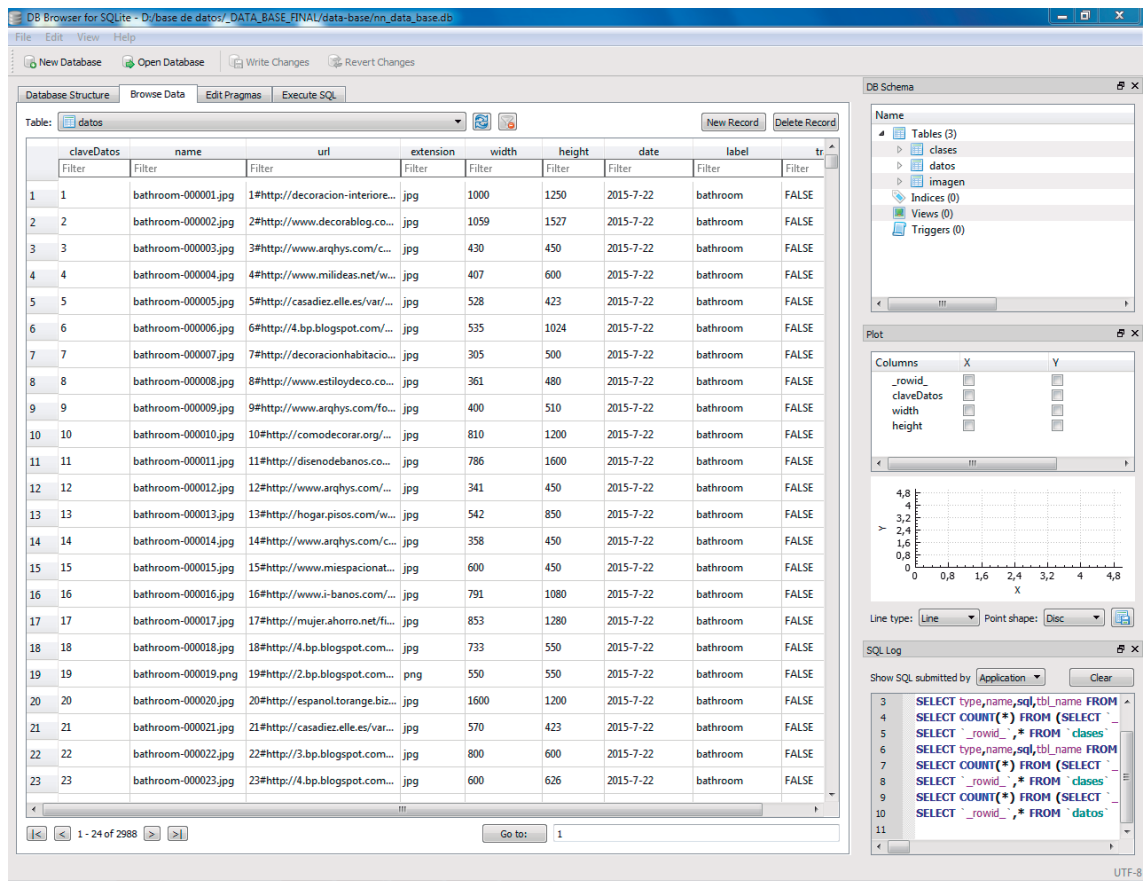


Figura 34. Captura de la base de datos en el gestor gráfico.

Programas de acondicionamiento de la base de datos y entrenamiento de la red neuronal convolucional en MATLAB

Para poder acceder a las imágenes es necesario pasar la base de datos antes creada a un tipo de datos que se pueda procesar, para ello utilizamos la librería de SQLite, esta vez compatible con MATLAB para poder pasar la base de datos a un conjunto de 3 estructuras (struct) donde se guardará cada una de las tablas. Posteriormente se recorren y ordenan para poder crear la misma base de datos en un solo struct con los datos necesarios como son el nombre, su ruta, su etiqueta, y sus dimensiones.

Para poder llevar a cabo la aplicación de data augmentation a las imágenes de la base de datos, se han creado varias funciones para realizar cada una de las variaciones a las imágenes recordando que son realizar un recorte aleatorio de porciones del 75% de la imagen, girarla, y corregir su aspecto de ratio.

Al final se dan valores de entrenamiento o validación a cada imagen en función de la tarea que va a ser incluida, en la Figura 35 y Figura 36 se muestra su diagrama de bloque con una explicación más detallada en el Anexo 5 y Anexo 3.

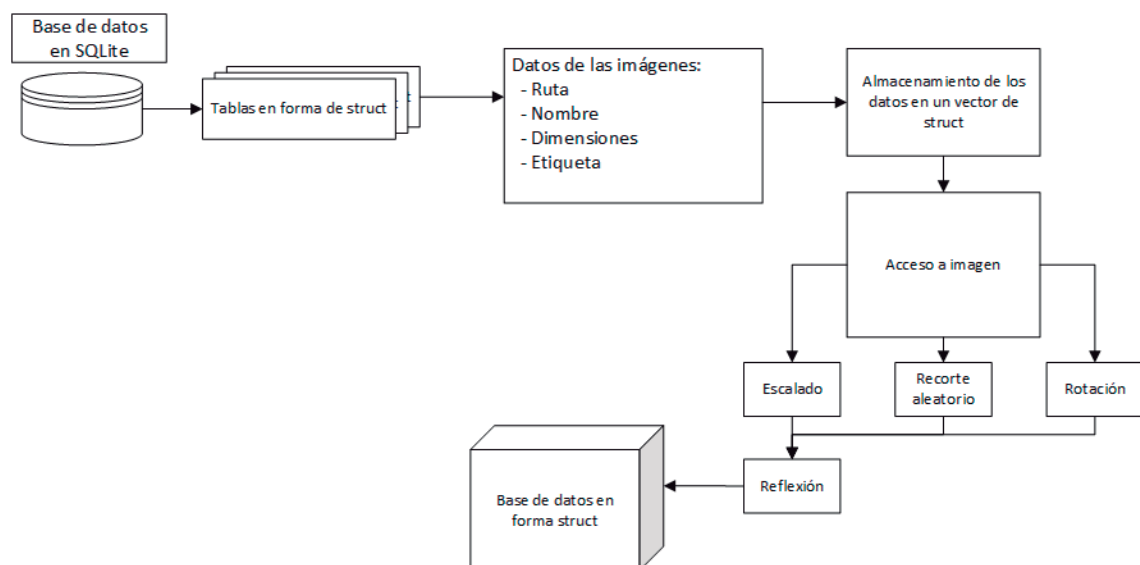


Figura 35. Diagrama de bloques del acondicionamiento de la base de datos.

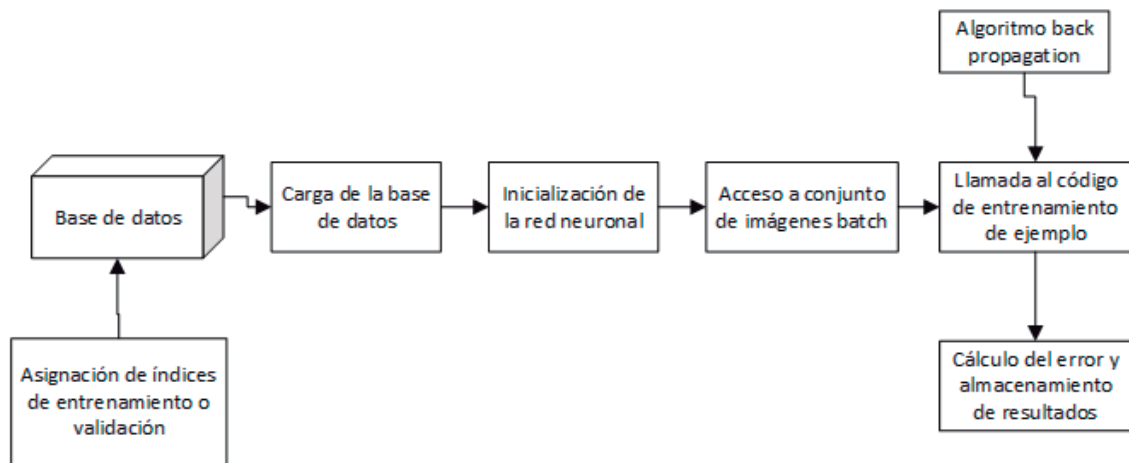


Figura 36. Diagrama de bloques del código que entrena la red neuronal.

Conclusiones y líneas futuras

Conclusiones

En este trabajo de fin de grado se han abordado algunas de las problemáticas que dificultan el uso de un sistema de Deep Learning orientado al caso concreto de reconocimiento de escenas dentro de un hogar y ofreciendo su potencial en otros ámbitos de la visión por computador.

Se ha conseguido obtener de forma sencilla y automatizada la búsqueda y recolección de imágenes utilizando como herramienta un potente buscador en Internet como es Bing, en una era donde la información está disponible fácilmente gracias a este tipo de servicios.

Se ha realizado con éxito la creación automática de una base de datos con un gran número de imágenes resolviendo la farragosa tarea de etiquetar y obtener las imágenes de las clases que se vayan a necesitar resumiéndola en la ejecución de un sencillo programa y su posterior comprobación.

Se ha realizado una labor de procesamiento de imagen aprovechando las capacidades que ofrece un entorno de programación destinado a ello como es MATLAB, siendo propuestas algunas formas de realizar el método de augmentation data para evitar ciertas prácticas que parecen poco eficientes enfocándose en el objetivo final.

Además se ha realizado un estudio de algunas decisiones que se pueden tomar a la hora de entrenar una red neuronal convolucional y sus consecuencias, intentando identificar posibles errores durante el entrenamiento que a simple vista pueden pasar desapercibidos pero que su detección a tiempo puede ahorrar tiempo y esfuerzo a la hora de resolverlos.

Se ha intentado entrenar una red neuronal convolucional para el reconocimiento de escenas dentro de una casa que ha resultado poco satisfactorio debido a un error en la inclusión de los datos a la red neuronal siendo posible obtener mejores resultados realizando un nuevo entrenamiento teniendo en cuenta el error cometido. Dado que el tiempo necesario para poder entrenar el sistema es demasiado grande, se espera poder corregir el error encontrado en el aprendizaje durante el trabajo y poder ofrecer resultados mejores en la presentación del mismo.

Líneas futuras

Puesto que no se han conseguido obtener resultados óptimos como fin último, si se han solucionados algunos aspectos importantes para este tipo de problemática, además se plantean algunas mejoras que podrían hacer más atractivo el potencial de este tipo de sistemas.

- **Estudio estadístico de las imágenes:** Estudiar las características estadísticas de las clases que se tienen para poder utilizar ese conocimiento a la mejora de la adaptación de los filtros o una implementación más eficiente de los cálculos.
- **Utilización remota de la base de datos:** Sería interesante poder crear, acceder y compartir las imágenes de una base de datos remotamente evitando así la necesidad descargar todas las imágenes y preprocesarlas siempre que se quiera realizar un trabajo de este tipo. Además de evitar manejar la gran cantidad de información, 24.5 GB en este trabajo, que necesitaría periféricos y equipos capaces, contribuiría a la mejora en las redes neuronales tanto en arquitecturas como en los cálculos necesarios.
- **Implementar el sistema en un lenguaje de tiempo real:** La implementación de una red neuronal en lenguajes capaces de procesar las imágenes en tiempo real llevarían funcionalidades de reconocimiento, detección y clasificación de objetos a tareas orientadas en aplicaciones de vídeo, campo cada vez más en auge gracias a los avances tecnológicos que se están dando.
- **Implementación remota:** Algo que no supondría demasiado esfuerzo ya que el uso de servidores y centros de cálculos es muy frecuente hoy en día, es posible aprovechar las posibilidades que ofrece un sistema de aprendizaje, orientado a la visión por computador, para acercar dispositivos como smartphones o equipos estándar como portátiles que no son capaces de llevar a cabo el paso de la imagen por la red neuronal pero que pueden mantener una conexión y transmisión de imágenes con soltura.
- **Mejora el filtrado de las imágenes de la base de datos:** una tarea que ha supuesto gran esfuerzo ha sido el descarte de imágenes que no serían útiles tanto por su composición como el hecho de estar repetidas en diferentes proporciones indetectables para el programa que se ha creado en este trabajo.

Referencias

- [1] Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks.
- [2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional.
- [3] *OpenCV*. (2015, Febrero). Retrieved from <http://opencv.org/>
- [4] *SQLite Home Page*. (2015, Febrero). Retrieved from <https://www.sqlite.org/>
- [5] *MatConvNet: CNNs for MATLAB*. (2015, Junio). Retrieved from <http://www.vlfeat.org/matconvnet/>
- [6] nVIDIA. (2015, Febrero). *Procesamiento paralelo CUDA*. Retrieved from <http://www.nvidia.es/object/cuda-parallel-computing-es.html>
- [7] Rosenblatt, F. (1961). *Principles of neurodynamics. Perceptron and the theory of brain mechanisms*.
- [8] Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *International Conference on Machine Learning*.
- [9] Welch, S. (2014, Noviembre). *Neural Networks Demystified*. Retrieved from <https://www.youtube.com/watch?v=bxe2T-V8XR8>
- [10] Vevaldi, A., & Lenc, K. (2014). *MatConvNet Convolutional Neural Networks for MATLAB*.
- [11] Cireşan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column Deep Neural Networks for Image Classification.
- [12] Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., & LeCun, Y. (2009). What is the Best Multi-Stage Architecture for Object Recognition? *International Conference on Computer Vision*, (pp. 2146–2153).
- [13] Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., . . . Bengio, Y. (2013). *Pylearn2: a machine learning research library*.
- [14] Vedaldi, A., & Zisserman, A. (2015, Julio). *VGG Convolutional Neural Networks Practical*. Retrieved from <http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html>
- [15] Quattoni, A., & Torralba, A. (2009). Recognizing Indoor Scenes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Anexo 1. Programación de descarga de imágenes en Bing utilizando Java

El programa consta de tres clases, *searchImage* que contendrá el método *main* del programa, *URLImage* que será el objeto que contenga información de la URL que se está utilizando y *DescargalImage* que será un thread que realice la descarga en paralelo de un conjunto de imágenes encontradas.

Clase *searchImage*

Se empieza obteniendo la ruta raíz en la que se está ejecutando el programa que servirá como punto de referencia para el guardado de las imágenes, abrirá y realizará una lectura, con un bucle *while*, línea a línea el fichero donde se encuentren las palabras a buscar, para poder diferenciar la búsqueda de la clase donde guardar las búsquedas basta con anteponer una doble barra (//) a las clase y después las palabras que buscar en cada línea, como se escribe a modo de ejemplo, los signos de suma (+) se utilizan para expresar espacios dentro de la búsquedas.

```
//kitchen  
cocina  
cocina+moderna  
cocina+rustica  
cuisine  
cuisine+moderne  
kitchen  
modern+kitchen  
cusina  
cusina+moderna
```

En cada palabra entrará en un método *realizarBusqueda* que comprueba que las carpetas Descargas, Fuentes y la carpeta, donde se guardarán las imágenes, llamada de la misma forma que la clase a la que pertenece.

Se crea el fichero donde se guardará el código fuente que devolverá el buscador, con su correspondiente *FileWriter* y *PrintWriter* asociados; después se crea la dirección URL a partir de la palabra, en una variable de tipo URL, que ha leído en el fichero de búsquedas, siguiendo el ejemplo anterior tomaría la siguiente forma para la primera

<http://www.bing.com/images/search?q=cocina&first=1>

Se realiza un *openStream* a la variable URL, para abrir una conexión a la dirección URL, asociando un *InputStreamReader* y un buffer al mismo para poder leer la respuesta de la conexión. Esta respuesta será el código fuente donde se buscarán las direcciones, se escriben en un fichero dentro de la carpeta Fuentes desde el buffer creado.

Se procede a la búsqueda de las URL donde están alojadas las imágenes con un método llamado *encontrarUrl*, al que se le pasan como argumento el fichero fuente y un fichero temporal que servirá para resolver problemas posteriores; se crean variables *FileReader* y *BufferedReader* para poder leer el fichero fuente y *FileWriter* junto a su *PrintWriter* para poder escribir en el fichero temporal.

La cabecera que tiene cada URL ha tenido que ser encontrada manualmente antes de la ejecución del mismo, siendo la de inicio *class="item"><a href="* y para la final *class="thumb"*, es posible que posteriormente sean diferentes debido a cambios en las páginas de los buscadores habiendo sido *img:"*; en el inicio del trabajo. Después se escribirán en el fichero temporal y volverá al método del que procedía.

Dado que por cada búsqueda que se realiza el buscador devuelven 35 imágenes se tiene que realizar varias peticiones hasta que devuelva cierto número de direcciones repetidas; cada respuesta a la petición contiene nuevas imágenes con otras repetidas de la petición anterior para dar continuidad a la búsqueda (se puede apreciar cuando se realiza una búsqueda y se va bajando con el ratón) que dependerá del tamaño de las imágenes por tanto se ha creado un método *corregirRepetidos*, que creará un vector de clases *URLImage* donde se guardará la URL y si ha sido repetida o no, de forma que se realizará una comparación entre los que tiene almacenados en el fichero temporal descartando las repetidas y escribiendo los nuevos en otro fichero Resultados teniéndolos en los dos de forma que en las siguientes búsquedas se mantienen las direcciones y las nuevas en el fichero resultado y en únicamente las nuevas que se tengan de descargar en el fichero temporal, así si se ejecuta la búsqueda en otro momento para actualizar las imágenes sólo descargará las nuevas.

Una vez encontradas las imágenes que se tengan que descargar se ejecuta el método *descargarImagenes* que se repartirán las direcciones entre el número de threads (*DescargalImage*) que se tienen coincidiendo con el número de procesadores que tenga el equipo y los pone en marcha realizando un *join* para esperar a que finalice cada uno devolverá el valor de imágenes descargadas para ser escrito por pantalla al método principal que pasará a la siguiente búsqueda o a la siguiente clase en caso de que el fichero de peticiones no haya finalizado, en su defecto finalizará el programa.

Clase *DescargalImage*

Se crean variables privadas donde guardarán la dirección URL, la ruta de la carpeta donde almacenar las imágenes, un identificador asignado para cada URL que servirá para dar nombre a la imagen, la etiqueta de su clase y tiempos de conexión y lectura.

Por cada dirección URL se crea un enlace a la dirección esperando como respuesta los bytes de la imagen que se recogen en un método *recibirBytes* que creará el archivo donde se va a guardar, la imagen, y va escribiendo la respuesta almacenada en el buffer, se tendrán en cuenta dos posibles errores, si se corta la comunicación antes de que haya acabado el fichero se modificará el tiempo de conexión previamente establecido, si bien lo que pasa es que no continúa la transmisión de los datos aun estando la conexión establecida se cerrará y se creará nuevamente un tiempo de lectura mayor, se realizarán hasta 3 intentos por imagen.

Clase URLImage

El objeto contendrá dos variables que serán la dirección y si ha sido repetida la dirección; además tendrá cuatro sencillos métodos, *getUrl*, *setUrl*, *getRepetido*, *setRepetido*, que servirán para dar y leer valores por cada una.

Anexo 2. Programación de creación y etiquetado de una base de datos

Para realizar el guardado y etiquetado de las imágenes, se utiliza una estructura jerarquizada de las clases a través de carpetas, habrá una carpeta principal de descargas donde se habrán guardado todas las imágenes organizadas en subcarpetas según su clase.

El programa en C++ realiza una comprobación de la base de datos y sus tablas, en caso de no tenerlas se crearán, a través de funciones ofrecidas por la librería SQLite [4], creadas las 3 tablas necesarias para poder identificar posteriormente a las imágenes se llevará a cabo el relleno de las tablas recorriendo las diferentes carpetas que componen el total de imágenes descargadas.

Se encontrará una ruta raíz donde reside el programa que se está ejecutando y a partir de ella se irán completando las rutas debidamente en función de la clase que se esté tratando realizando una búsqueda de todas al inicio para poder acceder a ellas a través de un vector de clases llamadas *jerarquiaCarpetas* y que contendrán los nombres de todas las carpetas dentro de la carpeta principal de descarga, o lo que es lo mismo, todas las clases que se van a tratar.

Con un bucle *for* se recorrerán las componentes de dicho vector, en cada componente se realizará el guardado de la clase en la tabla clases y se realizará la búsqueda y una llamada a la función *rellenarTablas* a la cual se le pasan como argumentos la ruta raíz y la clase, además de una puntero al archivo de la base de datos de tipo *sqlite3* propio de la librería.

Dentro de la función basta con recorrer la carpeta encontrando las imágenes, cada vez que encuentre una compondrá su ruta y se dispondrá a abrirla con la función *imread* perteneciente a la librería de OpenCV, si el acceso es válido se obtendrá un objeto de tipo *Mat*, también propio de OpenCV, que contendrá las propiedades de la imagen como son las dimensiones, el formato de la imagen entre otros, además de los bits que la componen.

Una vez obtenidos los datos que nos convengan se guardarán en una clase *Imagen* creada para guardar las propiedades de la imagen, pero sobre todo para dar transparencia al código ya que dentro se tienen funciones que tratan los cambios del tipo de variable que necesitan y devuelven las diferentes librerías, así si se realiza con otras librería basta con modificar o sustituir la clase con la conveniente.

Además de los detalles de la imagen se realiza una búsqueda de la URL en los ficheros creados en el programa de descarga en Java pudiendo ser inhabilitado este proceso, la búsqueda se realiza recorriendo el archivo y buscando un número identificador que se le fue asociado durante la descarga y que coincide con el identificador que contiene la imagen en su nombre.

Finalmente se llamará a la función *rellenarColumnas* la cual recibirá el objeto *Imagen* y el puntero a la base de datos rellenando uno a uno los campos pertinentes.

Una vez realizada una clase pasará a la siguiente siendo cerrada la base de datos cuando finalicen todos las clases.

Cabe mencionar que para cada lectura y escritura en la base de datos se debe realizar una petición y una espera mediante funciones de la librería disponibles en su página de documentación [4].

Anexo 3. Creación de código para aprovechamiento de un código ejemplo

El código del entrenamiento de la red neuronal se ha realizado a partir de la unión de dos códigos, por una parte el código de ejemplo de una red neuronal con la base de datos Imagenet que viene adjunto en la librería de MatConvNet [5], en ella se establece el orden que se debe seguir para el entrenamiento expresado en el siguiente esquema:

cnn_imagenet

- Inicialización de los campos a utilizar
 - tamaño del batch
 - uso de GPU
 - tasa de aprendizaje
 - número de subbatches por si no se tiene suficiente memoria
- Inicialización de la base de datos
- Inicialización de la red neuronal convolucional
- Cálculos de media y covarianza
- Establecimiento de la función de acceso a las imágenes *nn_get_batch*
- Llamada a función de entrenamiento, *cnn_train*, se le pasarán como argumentos
 - La red creada
 - Los índices de las imágenes destinadas al entrenamiento
 - La base de datos
 - Un puntero a la función de acceso a las imágenes
- Dentro de la función se realiza una actualización de los parámetros establecidos al inicio
- Se escogerá un batch, porción de imágenes, destinados al entrenamiento
- Se llama a la función *vl_simplenn* pasándole las imágenes además de parámetros de interés como la red neuronal o si resultados de anteriores pasos por la red, en ella se realiza:
 - Paso por las diferentes capas de la red neuronal
 - En caso de que sea hacia delante devolverá la predicción de la red
 - En caso de ser hacia atrás se pasará el gradiente como argumento y realizará el paso hacia atrás devolviendo los diferentes gradientes
- Una vez haya devuelto un valor se calculará su error o se actualizarán los pesos

- Por último si se ha llegado al final de la iteración se guarda la red entrenada hasta el momento y una gráfica en pdf que puede recuperarse en la información guardada en la red.

Ya que el orden de proceder es el mismo se decidió crear las funciones necesarias para que sea compatible el código anterior con la base de datos creada habiendo creado las siguientes funciones.

nn_net_init: realizará la inicialización de la red neuronal con la estructura que convenga, siguiendo como guía la función ya existente de la inicialización de la red neuronal se realiza

- Establecimiento de parámetros de la red neuronal
 - o Tamaño de imágenes
 - o Tipo de interpolación
 - o Asignación de la media de la base de datos previamente calculada
- Creación ordenada de las capas creando un cell array con structs dentro de cada componente dando los valores que necesitamos en cada caso explicado en el Anexo 4.

Como se ha mencionado a lo largo de la memoria, la diferencia de base de datos hace necesaria la creación de una función de acceso a las imágenes

nn_get_batch: se le pasará un conjunto de índices al igual que la base de datos

- Se creará el espacio necesario para una ejecución óptima del código
- Se recorrerá los índices seleccionados accediendo a la ruta de la imagen
- Se abrirá la imagen y en caso de error se creará una aleatoria artificial
- Se restará la media de la base de datos alojada en la misma
- Se guardará en un vector de 4 dimensiones donde las 3 primeras serán los canales de la imagen y la cuarta el número de imagen

Por tanto la unión del código ya creado con las otras dos funciones realizará el entrenamiento de la red neuronal.

Anexo 4. Formato de una red neuronal en MATLAB

La red neuronal será una composición jerarquizada de variable de tipo *struct*, *cell* y numéricos como *uint8* o *double*.

Las capas se guardarán en un *cell array* donde cada componente será un *struct* con el tipo de subcapa en cada momento, de forma que según el tipo que sea la capa en cada momento, especificado en todas las componentes del *cell array*, realizarán una función u otra. Además, cada subcapa, contendrá los diferentes parámetros que necesite siendo especialmente mencionables los siguientes:

Subcapa convolucional:

- **Weights:** será un *array cell* de dos componentes el primero guardará los pesos de los filtros y el segundo el vector de bias, en caso de no utilizar este último se creará vacío.
- **Stride:** indicará el paso con el que se filtrarán las imágenes
- **Pad:** indica el rellenado de ceros que necesita la imagen para poder coincidir en dimensiones al ser filtrado.
- **learningRate:** establecerá un factor de escala por el que tendrá que ser multiplicado la tasa de aprendizaje, se utilizan dos uno para los filtros y otro para el bias.
- **weightDecay:** marcará el decaimiento de los filtros y del bias.
- **Momentum:** establece el factor de momentum que marca la fiabilidad de los filtros en tiempos anteriores

Subcapa pool:

- **method:** método como se realiza el pooling pudiendo ser *máx* para coger el término de mayor valor o *avg* para realizar una media.

- **Stride:** indicará el paso con el que se realiza el pooling.
- **Pad:** indica el relleno de ceros que necesita la imagen para que no se produzcan problemas.
- **pool:** se expresa la dimensión de la matriz de pooling o dicho de otra forma, el número de píxeles adyacentes que se van a tener en cuenta.

Subcapa normalización:

- **param:** se establecen los parámetros N, κ, α, β para poder realizar la normalización de canales, se representa matemáticamente estos parámetros en el Anexo 5.

Subcapa dropout:

- **rate:** se especifica la probabilidad con la que se va a realizar la desactivación de una neurona.

Se tendrá en otra *struct* llamado normalization donde se tendrán el tamaño de las imágenes de entrada, el método de interpolación en cada de realizar un escalado, la imagen media de la base de datos.

Anexo 5. Creación de código para realizar data augmentation

Para la realización de data augmentation utilizada para la obtención de más imágenes, se han creado una función principal *nn_artificial_images* que contendrá otras explicadas a continuación

nn_crop: realizará el recorte de porciones de la imagen aleatoriamente

- Se calcula el porcentaje de la imagen original que se va a mantener multiplicando el número de sus filas y columnas por ese factor
- Se creará una rectángulo de selección donde se elegirá aleatoriamente píxeles que sirvan de referencia, serán las esquinas a partir de la cual se creará el rectángulo de corte
- Se selecciona parte de la imagen tomando como índices desde el de referencia hasta el número de filas y columnas deseados
- Se repetirá el proceso tantas veces como imágenes se deseen
- Se guardarán las imágenes en una estructura que devuelve

nn_rotate_image: realizará la rotación de la imagen en función de los ángulos pasado como argumento interpolando la posición de los bordes para aprovechar al máximo la imagen partiendo desde el centro

- Se quita el cero como grado de giro y se realiza a la imagen gracias a la función *imrotate*
- Se crea una máscara, matriz de unos, y se le realiza el mismo giro
- Se llama a la función *nn_find_corners_v2* que realiza las siguientes tareas
 - Obtiene el centro de la máscara girada aprovechando que tiene dimensiones impares
 - Se crean intervalos de búsqueda dividiendo en dos la imagen concéntricamente
 - Se calcula el área de una de las zonas de búsqueda sumando todas sus componentes
 - Si el resultado es igual al producto del número de filas y columnas de dicha zona no se ha encontrado el borde de la máscara y se realiza la misma operación cambiando el tamaño del área de búsqueda

- Si el resultado es diferente significa que han entrado ceros en la matriz y por tanto el borde estará en dicha área, se realizará el mismo proceso para la nueva zona
- Se realiza el proceso hasta que debido a restar el tamaño de la zona, éste sea negativo
- Se guardará la imagen en una estructura que se devuelve

nn_resize: se resuelve el problema del ratio de aspecto variable

- Se comprueba que el ratio de aspecto es cuadrado y se hace un reescalado convencional
- Si no es cuadrado se escoge el lado de menor tamaño, se crean los índices de forma que se cree una matriz cuadrada y se selecciona de la imagen
- Posteriormente se aplica el reescalado común
- Se repite el proceso tantas veces como imágenes cuadradas pueda contener la imagen original, en general suelen ser dos con un solapamiento entre las dos

Por tanto, la función principal llamará a las dos primeras funciones y posteriormente generará la reflexión en eje horizontal dando la vuelta a los índices de las columnas de la imagen; por último llamará a *nn_resize* para solventar posibles problemas de tamaños y guardará todo en una estructura que contendrá la misma forma de la base de datos original creando otra base de datos que será la utilizada en el trabajo.